BREAKING THE FACTORIZATION BARRIER IN DIFFUSION LANGUAGE MODELS WITH EXTERNAL MEMORY

Anonymous authors
Paper under double-blind review

ABSTRACT

Masked diffusion language models generate tokens in parallel, but factorized reverse steps cannot represent dependencies among simultaneously decoded tokens. At a fixed conditioning state, the KL divergence from the true joint conditional to the product of its exact marginals equals the block's total correlation. We introduce R-Joint, a training-free decoding framework that injects joint token structure into frozen diffusion language models through an editable external memory. Retrieved clean text spans define an empirical joint distribution over masked token blocks. We combine this distribution with the model's factorized proposal through a closed-form KL barycenter, with exact normalization using the sparse retrieval support. A gate tuned on validation text selects blocks for joint commitment. Experiments on LLaDA-8B and Dream-7B across Wikipedia, Python code, legal opinions, and synthetic entity corpora show consistent improvements in block exact match and likelihood at block length two. R-Joint improves exact match by up to 1.6 percentage points on global real-world corpora and by up to 3.0 points with repository-local memory. On Dream-7B, gains reach 6.8 points on synthetic entity spans and 9.5 points with denser memory. Matched ablations at the same block length show that preserving joint structure contributes beyond token-wise retrieval. At identical commit decisions, replacing the fused distribution with the product of its marginals reduces block exact match by 0.3 to 2.7 points. In iterative restoration of partially masked windows, joint commitment improves token accuracy with matching memory, even against a base decoder given more steps, and complements retrieved context. These results highlight the representational limits of factorized reverse steps and show that editable external memory can supply joint structure without retraining the model.

1 INTRODUCTION

Masked diffusion language models (dLLMs) generate text by unmasking positions round by round in a fully masked sequence (Austin et al., 2021; Lou et al., 2024; Sahoo et al., 2024; Shi et al., 2024). Open 7B–8B models such as LLaDA (Nie et al., 2025) and Dream (Ye et al., 2025) have made the paradigm practical, and so have commercial systems such as Mercury (Inception Labs et al., 2025). What sets them apart is parallel decoding. One forward pass proposes many tokens at once (Wu et al., 2025; Fang et al., 2026).

Parallelism carries a structural cost. A reverse step outputs one marginal distribution per masked position. Commit several positions in the same step and those marginals get multiplied, which assumes conditional independence. Each token can look plausible on its own while the tuple is still wrong as a whole. It may name the wrong entity, break an identifier or read as an inconsistent phrase. Section 2 makes the point precise. The KL divergence from the true joint conditional of a block to any product of per-position distributions is at least the block's total correlation. No per-position correction can close that gap, token-wise kNN interpolation in the style of kNN-LM (Khandelwal et al., 2020) included.

One line of prior work learns joint structure using auxiliary samplers, latent variables or low-rank tensors (Bansal & Sanghavi, 2025; Zhang et al., 2025; Shariatian et al., 2025; Kim & Sung, 2026). These approaches require training. Discrete Copula Diffusion (DCD) instead combines pretrained diffusion and autoregressive models at inference without fine-tuning (Liu et al., 2024). A second

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.

line coordinates the positions committed together (Kim et al., 2026; Ringel et al., 2026; Xia et al., 2026; Wang et al., 2026a; Yao, 2026). Most of these methods change only the commitment schedule. Yao (2026) also coordinates token values before commitment, using trained markers and an extra pass through the model's final layers. We supply the values from an external memory instead. It trains nothing and can be edited after deployment. A third line brings in external knowledge. BikNN adapts a non-autoregressive decoder with a datastore, but it builds its retrieval distribution per position (You et al., 2024). SARDI and ARAM retrieve documents into the context (Jünger et al., 2026; Kim & Ye, 2026). Table 2 in Appendix B lays out the gap. To our knowledge, no existing method uses an editable external memory to supply *joint* values for a block of a frozen dLLM.

Retrieved clean spans define an empirical tuple distribution, and it can carry dependencies the model's factorized proposal cannot. R-Joint puts it to work inside the reverse process. For a masked block, a noise-conditioned key retrieves complete clean spans from the datastore. Their weighted votes form a joint empirical distribution over the block. We fuse that distribution with the model's factorized proposal through a KL barycenter, which comes out as a normalized geometric mean. Exact normalization costs $O(kb)$, since the retrieval expert has at most k atoms. A gate calibrated on held-out raw text decides whether the fused top tuple is trustworthy. Blocks that pass are committed whole. The model weights never change. We tune two to four decoding scalars on validation text, once per corpus, block length and model. We evaluate on two model families and four corpora: code, Wikipedia, legal opinions, and a synthetic corpus of invented entities that neither model can have seen. Our contributions are as follows.[1]

- **Factorization barrier and a closed-form joint kernel.** Total correlation lower-bounds the factorized gap (Prop. 1, a standard identity). Our fused kernel is the closed-form KL barycenter, a logarithmic opinion pool (Prop. 2), and normalizes exactly on the sparse retrieval support (Prop. 3).
- **R-Joint, training-free joint decoding for frozen dLLMs.** It draws joint values for a block from an editable external memory. Clean n-gram coverage rises with every tested decade of corpus size, and hit rate tracks it at a roughly constant ratio, so retrieval improves with coverage (§4.1).
- **Controlled evidence that joint structure matters.** At block length two and fixed commit decisions, committing the product of R-Joint's own marginals instead of its joint tuple loses 0.3 to 2.7 percentage points (pp) of block exact match on every corpus and family in Table 5. A block is exact when all its tokens are correct. On natural text, per-token voting from the same neighbours trails R-Joint and has higher NLL. An arithmetic mixture raises NLL above the base there. Retrieved spans in the context recover about a third of the plain-vote synthetic gain (Appendix E).
- **Gains that track the knowledge gap and concentrate on names and numbers.** Block exact match follows the corpus ordering. It gains $+0.6$ pp on code, where base accuracy is highest, rising to $+5.2$ pp on invented entity spans and to $+7.7$ pp with denser memory (Table 1). The same ordering holds on Dream, which gains up to $+6.8$ pp on entity spans and $+9.5$ pp with denser memory (Figure 3, Table 5). On Wikipedia with LLaDA, on legal text and on repository code, blocks with a capitalized word or a number gain the most, $+2.0$ to $+3.2$ pp against corpus averages of $+1.0$ to $+1.9$ (Table 10). On repositories created after both models' training cut-offs, the family with the lower base accuracy gains more (Table 4).
- **End-to-end gains in iterative decoding.** Joint commitment improves per-position token accuracy on every corpus whose memory matches the query, even against a base decoder given about 40% more steps (Table 14). The gain is $+1.1$ to $+1.6$ pp on the synthetic corpus. Memory from other repositories gives only $+0.01$ pp, so the effect needs a memory that matches. Combined with retrieved context, joint commitment beats the context alone in token accuracy on every row of Table 14 and recovers fully masked entity mentions 1.5 to 1.6 times as often as the base decoder.

2 Problem: the factorized gap of parallel decoding

Setting. Let $x_0 = (x_{0,1}, \ldots, x_{0,L})$ be a clean token sequence over vocabulary $\mathcal{V}$. Let x_t be a partially masked copy, where each position is independently replaced by a mask token with probability ρ. A masked dLLM with frozen parameters θ maps x_t to one distribution per position, $p_\theta(v \mid x_t, i)$. A *block* is a set of consecutive masked positions $S = \{a, \ldots, a + b - 1\}$ with $b \in \{2, 3, 4\}$. Com-

[1] Code is available at `https://anonymous.4open.science/r/BREAKING-THE-FACTORIZATION-BARRIER-IN-DIFFU--SION-LANGUAGE-MODELS-WITH-EXTERNAL-MEMORY-56AA/`.

mitting the block in one step means sampling or maximizing the factorized proposal

$$p_\theta^{\otimes}(y_S \mid x_t) = \prod_{i \in S} p_\theta(y_i \mid x_t, i). \tag{1}$$

The gap is the total correlation. Let $P(y_S \mid x_t)$ be the true joint conditional of the block under the data distribution, and write P_i for its marginals. The best product distribution is their product, which need not equal $p_\theta^{\otimes}$. Even that best product differs from the true joint by a fixed amount.

Proposition 1 (Dependency gap). *For fixed x_t, $\min_{Q_1,\dots,Q_b} D_{\mathrm{KL}}\big(P(y_S \mid x_t) \,\big\|\, \prod_{i \in S} Q_i(y_i)\big) = \sum_{i \in S} H(P_i) - H(P) = \mathrm{TC}(S \mid x_t)$, the total correlation of the block (Watanabe, 1960). It is zero if and only if the block tokens are conditionally independent given x_t.*

Retrieval methods for language models improve the per-position marginals. Proposition 1 says that this cannot be enough. The bound is a minimum over all product distributions, so it holds even when every marginal is exactly right. What a per-position method cannot supply is the joint structure of the block, and that is the quantity R-Joint retrieves.

Appendix A gives the proof. For one conditioning state and one simultaneous commit, it also splits the divergence of the model's own proposal into $\mathrm{TC}(S \mid x_t) + \sum_{i \in S} D_{\mathrm{KL}}(P_i \,\|\, p_\theta(\cdot \mid x_t, i))$.

3 R-JOINT

R-Joint has four parts: a datastore of clean spans keyed by noisy context, a joint retrieval distribution, a closed-form fusion with exact normalization, and a calibrated gate for joint commitment. Figure 1 summarizes the pipeline.

3.1 A DATASTORE OF CLEAN SPANS UNDER NOISE

We build the datastore from raw text, with no labels. Documents are cut into windows of 512 tokens. For each window we sample a noise level $\rho \in \{0.15, 0.30, 0.50, 0.70\}$ and a random mask, then run the frozen model once on the masked window and harvest blocks from it. Inside each maximal run of masked positions we sample one block of every length $b \in \{2, 3, 4\}$ that fits, up to 18 blocks per window. The key for a block S is the average of the last-layer hidden states over S. We project that average to 256 dimensions with a fixed Gaussian matrix W and store it in half precision:

$$z_S = \Big(\tfrac{1}{b} \textstyle\sum_{i \in S} h_i\Big) W, \qquad W \in \mathbb{R}^{d \times 256} \text{ fixed.} \tag{2}$$

The value is the clean tuple $s_S = x_{0,S}$. Nothing is trained. One index holds all four noise levels because restricting retrieval to the matching noise bucket is worse in every cell of Table 3.

3.2 JOINT RETRIEVAL DISTRIBUTION

At decoding time the same function turns a candidate block S of x_t into a query key z. We retrieve the k nearest keys $\mathcal{N}_k$ of the same block length under Euclidean distance d_j. A softmax turns those distances into weights, at a data-dependent temperature equal to the median retrieved distance:

$$w_j = \frac{\exp(-d_j/\tau)}{\sum_{\ell \in \mathcal{N}_k} \exp(-d_\ell/\tau)}, \qquad r(y_S) = \sum_{j \in \mathcal{N}_k} w_j \, \mathbf{1}[y_S = s_j]. \tag{3}$$

Each neighbour votes for its complete tuple s_j. The per-token control of §4.2 instead builds one vote distribution per position and multiplies them. Write $\mathcal{U} = \{s_j : j \in \mathcal{N}_k\}$ for the set of distinct retrieved tuples, so $|\mathcal{U}| \le k$. We do not want the model forced to copy the datastore, so we smooth the retrieval distribution with the model's own proposal: $r^\epsilon = (1-\epsilon) r + \epsilon \, p_\theta^{\otimes}$ with $\epsilon = 0.1$.

Context-verified votes. Each entry also stores the three clean tokens on either side of its block. A query compares them with the candidate block's unmasked neighbours, and masked or out-of-window positions count as wildcards. We then re-weight neighbour j's vote by $\exp(\beta m_j)$, where m_j counts agreeing minus disagreeing context tokens. Setting $\beta = 0$ recovers (3). We choose β and k on the calibration split, along with the gate.

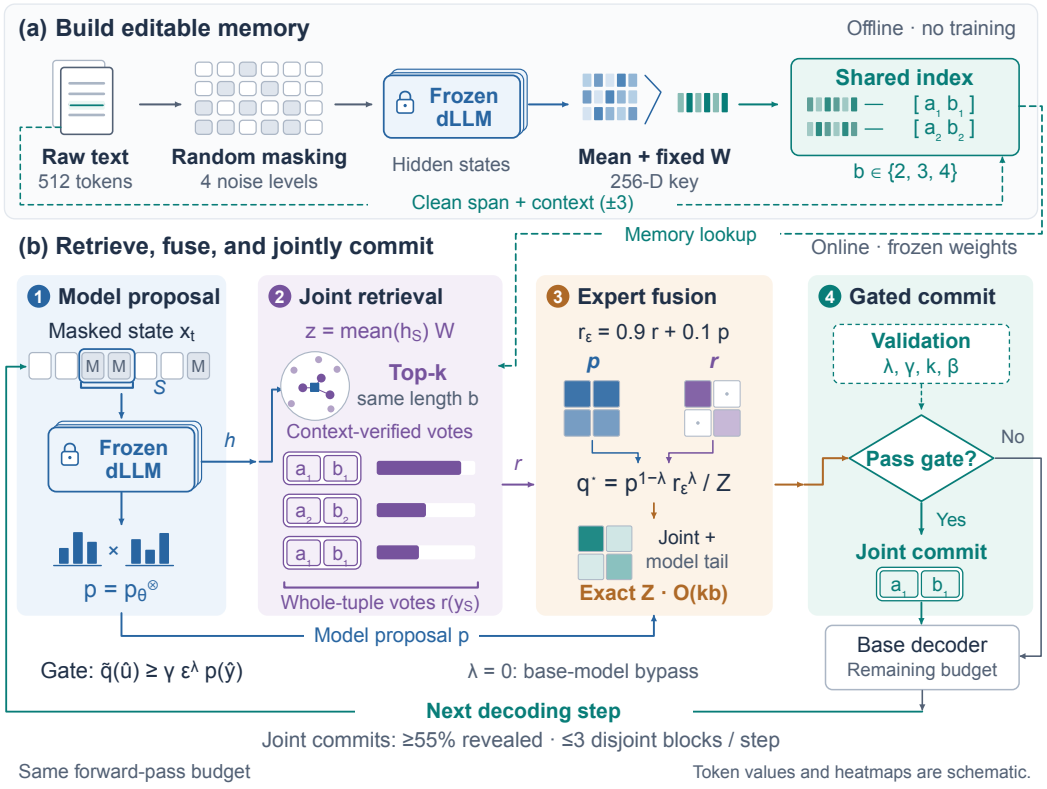

Figure 1: Overview of R-Joint. (a) Offline, frozen-model states from masked text index clean spans and context. (b) Online, context-verified tuple votes are smoothed and geometrically fused with the factorized model proposal. Exact $O(kb)$ normalization includes model-supported mass outside retrieval support. A calibrated gate selects joint commits. The base decoder fills the remaining step budget. Each reverse step is one forward pass, and retrieval reuses that step's hidden states rather than adding a pass.

3.3 Closed-form fusion and exact normalization

We fuse the two experts by the KL barycenter with trust $\lambda \in [0, 1]$:

$$q^\star = \arg\min_q \, (1-\lambda)\, D_{\mathrm{KL}}\big(q \,\big\|\, p_\theta^{\otimes}\big) + \lambda\, D_{\mathrm{KL}}(q \,\|\, r^\epsilon)\,. \tag{4}$$

Proposition 2 (Closed form). *For $0 < \epsilon < 1$ the objective in (4) is strictly convex and its unique minimizer is the normalized geometric mean $q^\star(y_S) = p_\theta^{\otimes}(y_S)^{1-\lambda}\, r^\epsilon(y_S)^\lambda / Z$.*

Normalization is exact on the sparse retrieval support and costs $O(kb)$, because the model-supported mass outside that support enters through a single closed-form term. Proposition 3 in Appendix A gives the constant Z and the exact sampling procedure.

3.4 Calibrated gating and joint commitment

We select a threshold on validation text. It decides whether a block is committed from the fused kernel or left to the model. Let $\hat{u} = \arg\max_{u \in \mathcal{U}} \tilde{q}(u)$ be the retrieved tuple with the largest unnormalized fused weight $\tilde{q}(u) = p_\theta^{\otimes}(u)^{1-\lambda} r^\epsilon(u)^\lambda$, and let $\hat{y} = \arg\max_y p_\theta^{\otimes}(y)$ be the model's own per-position argmax tuple. R-Joint commits $\hat{u}$ for the block if and only if

$$\tilde{q}(\hat{u}) \;\geq\; \gamma \cdot \epsilon^\lambda\, p_\theta^{\otimes}(\hat{y}), \tag{5}$$

and otherwise leaves the block to the model. The right-hand side is the fused weight the model's own tuple would get if the datastore had nothing to say about it. With $\gamma = 1$ the rule commits

only when the retrieved tuple dominates the model's choice even after that choice is discounted as unsupported. Setting $\lambda = 0$ is an explicit bypass. It leaves every block to the model whatever γ is, and reproduces the base model exactly.

Iterative decoding. R-Joint works inside an iterative decoder as a conservative add-on. Candidate blocks in masked runs that pass (5) are ranked by $\tilde{q}(\hat{u}) \cdot b$. Once 55% of the sequence is revealed, each step commits a non-overlapping set of at most three blocks. Committed positions count against the step's budget, so the number of forward passes does not change (Appendix G).

4 EXPERIMENTS

4.1 SETUP

Models. LLaDA-8B-Base (Nie et al., 2025) is the main model. Dream-v0-Base-7B (Ye et al., 2025) is the second family. Its logits at position i predict token $i + 1$, the convention of its Qwen initialization. We shift them accordingly and keep keys at block positions.

Corpora. For *Wikipedia* we use WikiText-103 (Merity et al., 2016). The training split supplies the datastore, validation is used for calibration, and the test split supplies queries. For *code* we use the `codeparrot-clean-valid` Python corpus (CodeParrot, 2021), split by document index into $94\%/3\%/3\%$. The *legal* corpus contains 21,186 New Hampshire court opinions from Pile of Law (Henderson et al., 2022), split by document into $90\%/5\%/5\%$. The *synthetic* corpus contains $12{,}000$ invented entities whose names are built from invented syllables, rendered with disjoint template sets for datastore, calibration and query text. Appendices E and H give its construction and the checks against leakage. Entity blocks are query blocks whose gold tuple contains an invented word.

Datastores and protocol. Each datastore is built from its corpus's training split in parts of up to $2{,}000$ windows. The main table uses one context-carrying build per corpus. Table 23 in Appendix H lists these builds and the context-free ones that other tables use. Block-level evaluation masks each query window with a random ρ, then samples one block per length inside each masked run, up to 18 per window, and runs one forward pass. Calibration uses $3{,}000$ windows on Wikipedia and code, $1{,}200$ on legal text, and $1{,}171$ on the synthetic corpus. Queries use the first 576 of the 590 Wikipedia test windows, for $6{,}005$ blocks of length 2. They use $1{,}500$ code windows, yielding $15{,}632$ such blocks, and $1{,}200$ windows each on legal text and the synthetic corpus. We use $\epsilon = 0.1$ throughout, and $k = 32$ unless stated otherwise or selected on calibration as in Table 1. The mask seed is 0 unless a seed replicate is stated. On code, dropping the query windows drawn from near-duplicate files leaves a calibrated gain of $+0.49$ pp, against $+0.56$ in Table 1 (Appendix H). We report block exact match (EM, %), block NLL in nats, and the residual repair rate. The last is the fraction of blocks that R-Joint fixes among those the base model gets wrong.

Coverage and retrieval scaling. Three quantities describe what a datastore can offer. The coverage ceiling counts how many query n-grams exist anywhere in the corpus. The hit rate is the share of query blocks whose gold tuple is among the retrieved values. The efficiency is its ratio to the share whose gold tuple is stored at all. Coverage rises with every decade of corpus size across the tested range. At about 10^8 tokens it reaches 89% of 2-grams on Wikipedia and 80% on code (Table 9). Along each hit-rate curve the efficiency stays roughly constant, at 75% on code and at 57% and 46% on Wikipedia bigrams and trigrams (Figure 4). So retrieval improves as coverage grows, and both quantities bound what the memory can offer. Appendix D describes how each quantity is measured.

4.2 JOINT VOTES VERSUS PER-TOKEN VOTES

Without gating, per-token votes from identical neighbours raise Wikipedia NLL ($3.40 \rightarrow 4.10$ at $\lambda = 0.5$), while joint votes improve every metric. At $\lambda = 0.9$, per-token votes cut Wikipedia EM by 12.8 pp to 36.7%, while joint votes (arm D) cut it by only 2.5 pp to 47.0% (Table 3). Independent votes can land on a tuple no neighbour ever voted for, and joint votes cannot.

Joint commitment against three calibrated alternatives. Per-token and mixture controls share neighbours, context-weighted votes and gate grids, but calibrate separately. The marginal control fixes R-Joint's commit decisions. At $b = 2$ the calibrated joint rule (Ccal in Table 5) has the

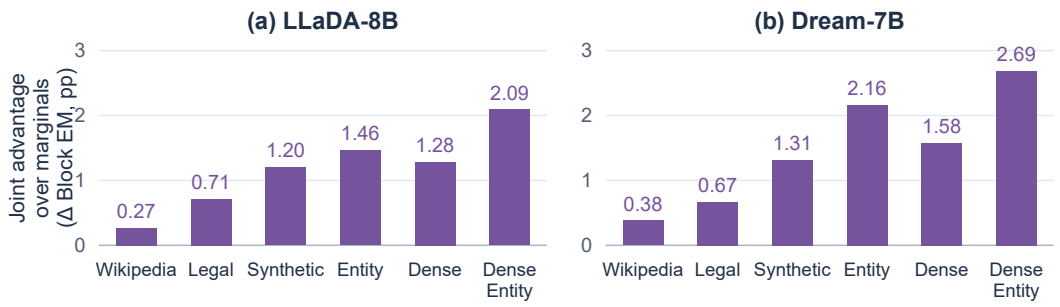

Figure 2: Joint tuple structure matters. At identical commit decisions ($b = 2$), R-Joint outperforms a matched control using the product of its own per-position marginals in block EM (Table 5). The largest gaps occur on synthetic entity spans. Dense denotes the denser synthetic memory.

lowest block NLL of any arm on all thirteen cells. Its exact match beats the marginal control on all thirteen, per-token voting on eleven and the mixture on nine. At identical commits, the product of its own marginals loses the least exact match on LLaDA Wikipedia (0.27 pp) and the most on Dream dense entity blocks (2.69 pp). Figure 2 plots this margin for every corpus but code, where it is 0.42 pp on LLaDA. Because neighbours, weights and commits are held fixed, the margin isolates joint structure, and it widens as the corpus moves from natural text towards invented entity spans. Its paired interval excludes zero in every cell but LLaDA Wikipedia. The mixture has higher exact match on Wikipedia and legal text, but there its NLL is above the base model. Once its calibration likelihood is constrained, it returns to $\lambda = 0$ on every natural corpus (MixN in Table 7). Tables 5, 6 and 7 give every cell, every interval and the scoring conventions.

4.3 Gains track the knowledge gap

With plain votes and $k = 32$, R-Joint improves block EM and NLL on every corpus (Table 1). Gains are smallest on code, where base accuracy is highest, and largest on invented entity spans. Natural text carries strong local signal, so a block's per-position marginals are often already right together. The headroom left for a joint memory is therefore smaller than on invented entity spans, where the model has no knowledge of the entity to fall back on. Figure 3 shows the same ordering across families. Calibrating k and the context weight β improves NLL throughout. Over plain votes it also raises EM on code, legal text and the synthetic corpus, and the largest EM gains appear where the knowledge gap is large. Table 5 gives the NLLs of both settings, with plain votes in column C0 and calibrated k and β in column Ccal. With calibrated k and β, gold-span probability on entity blocks rises $5.4\times$ over the base model. With the denser memory it rises $8.6\times$ and EM gains $+7.7$ pp.

The window-clustered 95% intervals of Table 5 exclude zero for every calibrated LLaDA gain, from $[+0.3, +0.8]$ pp on code to $[+6.9, +8.4]$ on dense entity blocks. On the natural corpora, document-clustered intervals keep the sign and a similar width. Three mask seeds agree (Appendix F). At $b = 3$ and $b = 4$ the calibrated synthetic gains stay positive, at $+1.2$ and $+0.4$ pp on LLaDA (Table 6).

The ordering transfers across families. With plain votes, Dream gains $+1.01$ pp on legal text, $+3.26$ pp on all synthetic blocks and $+4.64$ pp on entity blocks. With calibrated k and β, its gains follow the same ordering (Figure 3) and reach $+9.51$ pp on dense entity blocks. Table 5 gives these Dream cells, and Table 8 shows the gains of both families rising from compact to full datastores.

Table 10 splits plain-vote gains on the context-free builds and the 263-repository set of §4.4 by span type. Blocks with a number gain $+2.7$ pp on LLaDA Wikipedia, and blocks with a capitalized token gain $+2.0$ to $+3.2$ pp on legal text and repository code. Their corpus averages are $+1.0$ to $+1.9$ pp.

What the memory injects. We freeze the matched-calibration gate and swap in the memory of a second synthetic corpus. It is drawn with another seed and has the same templates but different entities. About half of the gain does not survive the swap. Matched memory beats this substituted memory by $+1.9$ pp on all blocks and $+2.7$ on entity blocks with LLaDA, and by $+2.4$ and $+4.0$ with Dream. All four paired intervals exclude zero. On LLaDA the gaps are 51% and 52% of the calibrated gains in Table 1. The substituted memory also changes surface forms and candidate

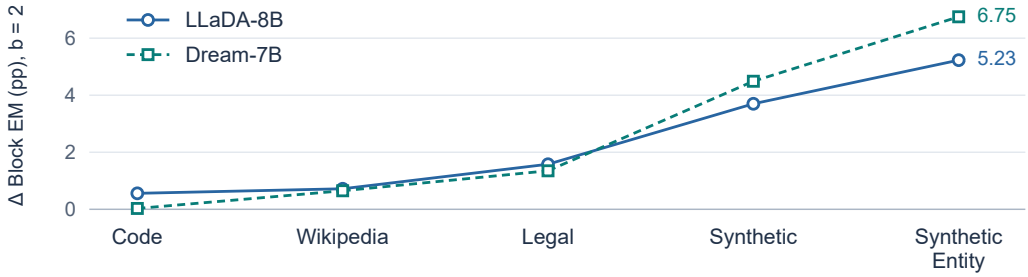

Figure 3: Gains increase from code to invented entity spans in both families. Points use calibrated k, β on context-carrying datastores (Tables 1 and 5). Dream code has no context-carrying build, so its point uses plain votes on the full code build of Table 12.

Table 1: Main result: R-Joint on four corpora with LLaDA-8B at block length $b = 2$, using the context-carrying datastores (Appendix H). We compare plain votes with $k = 32$ to votes with k and the context weight β selected on calibration. The last two columns refer to the calibrated setting. Gold prob. is $\exp(-\Delta\text{NLL})$, the geometric-mean ratio of gold-span probabilities. Repair is the fraction of base errors that this setting fixes. The same setting also turns 1.6% of base-correct blocks wrong on code, 2.5% on Wikipedia and 4.5% on legal text. Appendices B and E report other block lengths, the context-free builds and Dream-7B. Table 23 names the build behind every cell.

			plain votes, k=32		calibrated k, β			
Corpus	gap	base EM	EM	Δ (pp)	EM	Δ (pp)	gold prob.	repair
Code	smallest	70.00	70.53	+0.53	70.56	**+0.56**	×1.01	5.6%
Wikipedia	small	49.49	50.26	+0.77	50.21	+0.72	×1.08	3.8%
Legal	medium	47.47	48.62	+1.15	49.05	**+1.58**	×1.14	7.1%
Synthetic, all	large	35.83	38.58	+2.75	39.53	**+3.70**	×2.84	7.5%
Synthetic, entity	largest	14.08	18.16	+4.08	19.31	**+5.23**	**×5.35**	–
denser memory (24 templ.)	largest	14.08	19.12	+5.04	21.74	**+7.66**	**×8.64**	–

coverage, so the gap compares two memory conditions rather than isolating the facts. Appendix H gives all differences, the dense-memory margins and the intervals.

The key carries identity even when the whole name is hidden. We re-draw every entity's spouse and compare an edited memory, which holds the new spouses, with the stale one. Over all 1,129 edited-name blocks on LLaDA and 1,134 on Dream, the edited memory leads by $+0.4$ and $+0.2$ pp. Both intervals cover zero, because a visible fragment identifies the record either way. On the 136 and 138 blocks whose name is hidden in full with no visible cue, the base model scores 0.00 and the edited memory leads by $+5.9$ and $+5.1$ pp. These are eight and seven blocks, and both intervals exclude zero. A name-level probe over 1,194 fully masked target names gives a lead of $+6.03$ pp $[+4.69, +7.45]$ in name EM. Fully masked mentions are also recovered more often in end-to-end decoding (§4.5). Appendix H gives the edited-world cells and the name probe.

4.4 Local memory: repositories and articles

We study two local real-world memories, repository files and article paragraphs. Repository files are reported here, and article paragraphs add $+0.2$ to $+0.3$ pp (Appendix E). The block-level results of Table 4 use the post-cut-off repository set, while Table 10 and the end-to-end results of §4.5 use a set of 263 public repositories. We group by repository or article and hold out one file or paragraph in ten for queries, one in ten for calibration. The rest becomes the datastore, with windows that never cross a group, and retrieval stays inside the query's own group. The post-cut-off repositories were created after both models' stated cut-offs, which makes prior exposure unlikely. The caption of Table 4 in Appendix B gives the deduplication rule. LLaDA's base block EM on them is 75.0%, against 70.0% on the global code split (Table 1). Same-repository memory adds $+1.14$ pp, cuts block NLL by 7.2% and repairs 7.9% of its errors. Dream's base EM is 65.2%. Same-repository memory adds $+2.99$ pp at $b = 2$, cuts block NLL by 9.6% and repairs 9.1% of its errors. The gain

is larger for the family with the lower base accuracy. Memory from *other* repositories adds nothing on either family, with changes of $+0.04$ and -0.03 pp.

4.5 END-TO-END DECODING

We integrate R-Joint into an iterative decoder that restores masked windows in T confidence-ordered steps. We compare three uses of the same retrieval against a base decoder given about 40% more steps. These are joint commitment at $T = 16$, retrieved source windows written into the context with no fusion or gate (the prefix), and their combination. The base decoder uses $T = 22$ (Table 14, Appendix G). At $T = 31$ the base decoder costs at least as much per window as the combination. Its accuracy then changes by at most 0.04 pp on the five natural-corpus rows and the same-repository row (Table 18). On these rows the base decoder is therefore not step-starved at $T = 22$, and extra steps are no substitute for memory.

Joint commitment raises token accuracy on every row with a matching memory, and it helps most where the model lacks knowledge. Token accuracy rises by $+1.1$ pp on the synthetic corpus with LLaDA and $+1.6$ with Dream, and by $+1.0$ on repository-local code (Table 14). On natural text it rises by $+0.26$ to $+0.42$ pp across both families. Paired intervals over windows exclude zero on each of these cells. With retrieval restricted to *other* repositories the gain is $+0.01$ pp $[-0.16, +0.17]$, so the effect requires a memory that matches the query. Joint commitment also exceeds the prefix alone on LLaDA code, Dream legal text and both mentions-masked rows.

The two interventions are complementary. In token accuracy the combination leads the prefix alone on every row of Table 14. It adds $+0.27$ to $+0.34$ pp on the natural corpora and on repository-local code. On the synthetic corpus it adds $+0.30$ pp with LLaDA and $+0.45$ with Dream, and $+0.53$ and $+0.69$ with the denser memory. With memory from other repositories it adds only $+0.07$ pp. On mention exact match the combination leads the prefix in five of the six synthetic rows (Tables 14 and 20). When every mention is masked in full, the combination recovers 4.14% of them on LLaDA and 5.50% on Dream, against 2.63% and 3.76% for the base decoder. Appendix G gives the rates of all four arms. The combination costs at most 3% more time per window than the prefix alone and about 40% more than the base decoder at $T = 22$, because a prefix doubles the attended length of every pass after the first. Even at $T = 31$ the base decoder trails the combination by 0.44 to 2.06 pp on the natural-corpus and same-repository rows (Table 18). On those rows the ordering is therefore no artefact of unequal compute. Joint commitment on its own is the cheapest arm of the four. The combination is the dearest and also the strongest in token accuracy on every corpus we test. On whole masked runs of the LLaDA Wikipedia windows the three arms keep their token-accuracy ordering. Joint commitment raises the share of runs restored in full by $+0.41$ pp, the prefix by $+0.58$ and the combination by $+0.95$ (Appendix G). The comparison with the prefix measures the whole system, since the combination also consults the memory on every gate-open step while the prefix consults it once. To isolate the commitment rule we hold the retrieval schedule fixed and change only the committed value. The joint tuple beats the product of its own marginals, the Marg arm, by $+0.13$ to $+0.44$ pp. It beats the B arm, which votes per token over its own neighbours, by $+0.10$ to $+0.29$. Every paired interval over windows excludes zero (Table 16). Retrieved context and joint commitment act on different errors, so the memory is best used in both ways at once.

The step sweeps of Table 15 and the commit decomposition of Table 17 show how the gain arises. On Wikipedia, the base decoder's own ordered commitment raises token accuracy by 1.0–1.4 pp and the fraction of masked runs restored entirely by 2.3–2.6 pp from $T = 1$ to $T = 8$. On code it raises them by $+0.2$ and $+1.0$ pp. Joint commitment adds its gain on top of this ordering. At $T = 1$ and $T = 2$ the gate already commits about 600 and $1{,}650$ blocks and moves accuracy by at most 0.03 pp. From $T = 4$ upward the gain over the base decoder is positive on all three series we swept. At $T = 16$ it reaches $+0.39$ pp on LLaDA Wikipedia, $+0.32$ on Dream Wikipedia, and $+0.19$ on LLaDA code. From $T = 1$ to $T = 16$, commits grow from about 600 blocks at 87–94% precision to $8{,}740$–$9{,}200$ blocks at 39–51%, yet accuracy rises. A commit matters for its advantage over the position it replaces, rather than its precision. On the matching-memory rows of Table 17 the gate commits 19 to 25% of the masked tokens. On those positions it is $+1.3$ to $+1.8$ pp more accurate than the base decoder on natural text and $+4.6$ pp on repository-local code. Both arms decode with full feedback, so a joint commit also changes the context of the positions it leaves to the base ordering. We therefore report both terms of the commit decomposition per window. The uncommitted term is small, between -0.03 and $+0.07$ pp, and the two terms together reproduce the

end-to-end difference on every row. With memory from other repositories the committed advantage is $+0.12$ pp and the end-to-end difference disappears. Seed replicates, the neighbour count, and further controls are in Appendix F.

5 Related Work

Diffusion language models. Discrete and masked diffusion models generate text through iterative denoising over discrete token sequences, allowing multiple masked positions to be predicted within the same reverse step rather than following a strictly autoregressive order (Austin et al., 2021; Lou et al., 2024; Sahoo et al., 2024; Shi et al., 2024; Ou et al., 2024). Recent work has scaled this paradigm to open 7B–8B models, including LLaDA and Dream (Nie et al., 2025; Ye et al., 2025).

Joint dependencies and commit order. ADJUST, VMD, LADD and tensor-train models learn token dependencies through trained samplers, latent variables or tensor heads (Bansal & Sanghavi, 2025; Zhang et al., 2025; Shariatian et al., 2025; Kim & Sung, 2026). DCD instead combines pretrained diffusion and autoregressive models without fine-tuning (Liu et al., 2024). DAPD, DEMASK, DepCap and TACG select or schedule positions for parallel commitment (Kim et al., 2026; Ringel et al., 2026; Xia et al., 2026; Wang et al., 2026a). Others revise decoding states or earlier commitments, or hand blocks to a trained autoregressive model (Zhu et al., 2025; Wang et al., 2026b; Hersche et al., 2026). Yao (2026) trains LoRA adapters and marker vectors. After commit-set selection, it re-applies the final layers to coordinate the marked positions and then reads token values from the resulting logits. That work motivates joint commitment with the same conditional total correlation we use. All of them reach dependencies through the model. R-Joint instead reads joint values from an external datastore that can be edited or replaced without training.

External memory. Bi-kNN builds a datastore under several mask ratios for an iterative non-autoregressive decoder (You et al., 2024), but does so per position. Table 3 shows that per-position votes cannot carry the joint information missing from parallel decoding. SARDI and ARAM retrieve documents into a dLLM's context (Jünger et al., 2026; Kim & Ye, 2026). That changes the conditioning but keeps the per-position factorization. PoE-Bridge fuses a dLLM with an autoregressive expert (Shi et al., 2026) and draws dependencies from a second model rather than memory. kNN-LM and kNN-MT interpolate a per-token retrieval distribution into an autoregressive model (Khandelwal et al., 2020; 2021). NEST and SILO retrieve spans or isolate sensitive text (Li et al., 2024; Min et al., 2024). Autoregressive decoding only needs per-token distributions, whereas diffusion decoding lacks the joint distribution.

6 Limitations and Conclusion

Parallel decoding multiplies per-position marginals, so even exact marginals leave a gap equal to the block's total correlation (Proposition 1). No per-position correction can close it. R-Joint retrieves clean spans from an editable memory. A closed-form KL barycenter fuses their empirical joint distribution with the frozen model's proposal. Nothing is trained, and normalization is exact at $O(kb)$ cost. On two model families and four corpora, block EM rises by up to 1.6 pp on natural text with global datastores and 3.0 pp with repository-local memory (Tables 1 and 4). Invented entity spans gain 5.2 pp on LLaDA and 6.8 on Dream, and 7.7 and 9.5 with denser memory. Joint structure matters in its own right. At $b = 2$ and fixed commit decisions, the product of R-Joint's own marginals loses 0.3 to 2.7 pp of block EM (Table 5). In the decoder, at a fixed retrieval schedule, the joint tuple beats that product by $+0.13$ to $+0.44$ pp of token accuracy (Table 16). Joint commitment raises token accuracy on every corpus with matching memory, even against a base decoder given about 40% more steps. It does so at about three quarters of the cost. With retrieved context, it recovers fully masked entity mentions 1.5 to 1.6 times as often as the base decoder (Table 14).

Our end-to-end evaluation restores windows whose unmasked positions are real text. It measures iterative recovery, not generation of a fully masked answer from a prompt. The gain comes from matching memory content. With memory from other repositories, the committed positions gain only $+0.12$ pp and the end-to-end difference disappears (Table 17). These results establish joint dependency modeling as a structural limitation of parallel decoding, and editable memory as a training-free way to inject joint structure. Knowledge can then be updated by editing the datastore (§4.3).

AI USE STATEMENT

We used LLM-based coding assistants under the authors' direction. They helped write, debug and run the experiment code, search the literature, and draft and edit the manuscript. The authors set the research questions, the method, the experimental protocol, the reporting choices and the interpretation of every result. They reviewed the AI-assisted code and text, and verified the reported results. We take responsibility for the final content of this work, including text, claims and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work uses only public corpora. They are WikiText-103, an English Wikipedia dump, a public Python code corpus, public Python repositories, court opinions from Pile of Law, and a synthetic corpus of invented entities. No human subjects are involved. A retrieval datastore returns verbatim spans from its corpus, so a deployment must build the datastore only from text that may be reproduced. The same property makes the memory editable and auditable. Isolating sensitive text in a separate datastore is a natural safeguard (Min et al., 2024).

REPRODUCIBILITY STATEMENT

Section 4.1 and Appendix H give all protocol constants, including window length, noise levels, key construction, datastore sizes, calibration and query splits, and the context-verification rule. Appendix H also gives the decontamination procedure and counts for the enlarged datastores. Appendix A contains proofs of the three propositions and gives the full gate grid. Table 23 lists the datastore builds and the tables that use them. Appendix H lists the selected settings for the $b = 2$ main runs, and every cell uses mask seed 0 unless a seed replicate is stated. For the end-to-end runs, Figure 5 gives the decode loop of all four arms in the order it executes. Table 19 gives one window's execution trace. Table 21 gives a derived split of the measured time per window into model compute, retrieval, fusion and prefix construction.

Code is available at `https://anonymous.4open.science/r/BREAKING-THE-FACTORIZATION-BARRIER-IN-DIFFU--SION-LANGUAGE-MODELS-WITH-EXTERNAL-MEMORY-56AA/`.

The supplementary material provides the datastore builder, gated block evaluator, end-to-end decoder, synthetic corpus generator, paired bootstrap, the per-block outputs behind every interval, and the selected calibration settings.

REFERENCES

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured Denoising Diffusion Models in Discrete State-Spaces. *arXiv preprint arXiv:2107.03006*, 2021.

Parikshit Bansal and Sujay Sanghavi. Enabling Approximate Joint Sampling in Diffusion LMs. *arXiv preprint arXiv:2509.22738*, 2025.

CodeParrot. CodeParrot Dataset Cleaned (valid). `https://huggingface.co/datasets/codeparrot/codeparrot-clean-valid`, 2021. Hugging Face dataset `codeparrot/codeparrot-clean-valid`.

Liancheng Fang, Aiwei Liu, Henry Peng Zou, Yankai Chen, Enze Ma, Leyi Pan, Chunyu Miao, Wei-Chieh Huang, Xue Liu, and Philip S. Yu. Locally Confident, Globally Stuck: The Quality-Exploration Dilemma in Diffusion Language Models. *arXiv preprint arXiv:2604.00375*, 2026.

Christian Genest and James V. Zidek. Combining Probability Distributions: A Critique and an Annotated Bibliography. *Statistical Science*, 1(1):114–135, 1986. doi: 10.1214/ss/1177013825.

Peter Henderson, Mark S. Krass, Lucia Zheng, Neel Guha, Christopher D. Manning, Dan Jurafsky, and Daniel E. Ho. Pile of Law: Learning Responsible Data Filtering from the Law and a 256GB Open-Source Legal Dataset. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2022. arXiv:2207.00220.

Michael Hersche, Nicolas Menet, Ronan Tanios, and Abbas Rahimi. Locally Coherent Parallel Decoding in Diffusion Language Models. In *International Conference on Machine Learning*, 2026. arXiv:2603.20216.

Geoffrey E. Hinton. Training Products of Experts by Minimizing Contrastive Divergence. *Neural Computation*, 14(8):1771–1800, 2002. doi: 10.1162/089976602760128018.

Inception Labs, Samar Khanna, Siddhant Kharbanda, Shufan Li, Harshit Varma, Eric Wang, Sawyer Birnbaum, Ziyang Luo, Yanis Miraoui, Akash Palrecha, Stefano Ermon, Aditya Grover, and Volodymyr Kuleshov. Mercury: Ultra-Fast Language Models Based on Diffusion. *arXiv preprint arXiv:2506.17298*, 2025.

Paul Jünger, Justin Lovelace, Linxi Zhao, Dongyoung Go, and Kilian Q. Weinberger. Self-Augmenting Retrieval for Diffusion Language Models. In *International Conference on Machine Learning*, 2026. arXiv:2606.06474.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through Memorization: Nearest Neighbor Language Models. In *International Conference on Learning Representations*, 2020. arXiv:1911.00172.

Urvashi Khandelwal, Angela Fan, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Nearest Neighbor Machine Translation. In *International Conference on Learning Representations*, 2021. arXiv:2010.00710.

Bumjun Kim, Dongjae Jeon, Moongyu Jeon, and Albert No. DAPD: Dependency-Aware Parallel Decoding via Attention for Diffusion LLMs. In *International Conference on Machine Learning*, 2026. arXiv:2603.12996.

Byoungkwon Kim and Minhyuk Sung. Tensor-Train Joint Modeling for Few-Step Masked Diffusion. *arXiv preprint arXiv:2607.03788v2*, 2026. URL `https://arxiv.org/pdf/2607.03788v2`.

Jaemin Kim and Jong Chul Ye. Adaptive Guidance for Retrieval-Augmented Masked Diffusion Models. *arXiv preprint arXiv:2603.17677*, 2026.

Minghan Li, Xilun Chen, Ari Holtzman, Beidi Chen, Jimmy Lin, Wen-tau Yih, and Xi Victoria Lin. Nearest Neighbor Speculative Decoding for LLM Generation and Attribution. In *Advances in Neural Information Processing Systems*, volume 37, pp. 80987–81015, 2024. arXiv:2405.19325.

Anji Liu, Oliver Broadrick, Mathias Niepert, and Guy Van den Broeck. Discrete Copula Diffusion. *arXiv preprint arXiv:2410.01949*, 2024.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete Diffusion Modeling by Estimating the Ratios of the Data Distribution. In *International Conference on Machine Learning*, 2024. arXiv:2310.16834.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer Sentinel Mixture Models. *arXiv preprint arXiv:1609.07843*, 2016.

Sewon Min, Suchin Gururangan, Eric Wallace, Weijia Shi, Hannaneh Hajishirzi, Noah A. Smith, and Luke Zettlemoyer. SILO Language Models: Isolating Legal Risk In a Nonparametric Datastore. In *International Conference on Learning Representations*, 2024. arXiv:2308.04430.

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. *arXiv preprint arXiv:2502.09992*, 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. *arXiv preprint arXiv:2406.03736*, 2024.

Liran Ringel, Ameen Ali, and Yaniv Romano. Dependency-Guided Parallel Decoding in Discrete Diffusion Language Models. *arXiv preprint arXiv:2604.02560*, 2026.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and Effective Masked Diffusion Language Models. In *Advances in Neural Information Processing Systems*, 2024. arXiv:2406.07524.

Dario Shariatian, Alain Durmus, Umut Simsekli, and Stefano Peluchetti. Latent-Augmented Discrete Diffusion Models. *arXiv preprint arXiv:2510.18114*, 2025.

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis K. Titsias. Simplified and Generalized Masked Diffusion for Discrete Data. In *Advances in Neural Information Processing Systems*, 2024. arXiv:2406.04329.

Juntong Shi, Brian L. Trippe, Jure Leskovec, Stefano Ermon, and Minkai Xu. Diffusion Language Model Parallel Decoding via Product-of-Experts Bridge. In *International Conference on Machine Learning*, 2026. arXiv:2606.08048.

Chengcheng Wang, Tingzhang Luo, Wenhao Li, Jianyuan Guo, and Chang Xu. TACG: Trajectory-Aware Commit Gating for Diffusion Language Model Decoding. *arXiv preprint arXiv:2607.03236*, 2026a.

Xinyun Wang, Min Zhang, Sen Cui, Zhikang Chen, Bo Jiang, Kun Kuang, and Mingbao Lin. Reversible Diffusion Decoding for Diffusion Language Models. *arXiv preprint arXiv:2602.00150*, 2026b.

Satosi Watanabe. Information Theoretical Analysis of Multivariate Correlation. *IBM Journal of Research and Development*, 4(1):66–82, 1960. doi: 10.1147/rd.41.0066.

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 preprint arXiv:2505.22618*, 2025.

Xiang Xia, Wuyang Zhang, Jiazheng Liu, Cheng Yan, and Yanyong Zhang. DepCap: Adaptive Block-Wise Parallel Decoding for Efficient Diffusion LM Inference. *arXiv preprint arXiv:2604.15750*, 2026.

Lin Yao. Don't Commit Alone: Joint Token Commitment in Diffusion Language Models. *arXiv preprint arXiv:2607.04469*, 2026.

Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7B: Diffusion Large Language Models. *arXiv preprint arXiv:2508.15487*, 2025.

Wangjie You, Pei Guo, Juntao Li, Kehai Chen, and Min Zhang. Efficient Domain Adaptation for Non-Autoregressive Machine Translation. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 13657–13670, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.810. URL https://aclanthology.org/2024.findings-acl.810/.

Yichi Zhang, Alex Schwing, and Zhizhen Zhao. Variational Masked Diffusion Models. *arXiv preprint arXiv:2510.23606*, 2025.

Qinglin Zhu, Yizhen Yao, Runcong Zhao, Yanzheng Xiang, Amrutha Saseendran, Chen Jin, Philip Teare, Bin Liang, Yulan He, and Lin Gui. Latent Refinement Decoding: Enhancing Diffusion-Based Language Models by Refining Belief States. *arXiv preprint arXiv:2510.11052*, 2025.

A PROOFS

Proposition 3 (Exact normalization on sparse support)**.** *For $y_S \notin \mathcal{U}$, $r(y_S) = 0$ and the unnormalized weight is $\epsilon^\lambda p_\theta^{\otimes}(y_S)$. Hence*

$$Z = \epsilon^\lambda \Big[1 - \sum_{u \in \mathcal{U}} p_\theta^{\otimes}(u)\Big] + \sum_{u \in \mathcal{U}} p_\theta^{\otimes}(u)^{1-\lambda} \big[(1-\epsilon) r(u) + \epsilon\, p_\theta^{\otimes}(u)\big]^\lambda, \tag{6}$$

which costs $O(kb)$ to evaluate. Sampling from $q^\star$ is exact. First, choose between the retrieved tuples and the remainder according to their total mass. In the remainder, sample from $p_\theta^{\otimes}$ and reject tuples in $\mathcal{U}$.

We select the scalars (λ, γ) separately for each corpus, block length and model on the calibration split. For the calibrated setting of Table 1 (Ccal in Appendix B), we also select the neighbour count $k \in \{32, 64, 128\}$ and the context weight $\beta \in \{0, 0.5, 1, 2\}$. The full grid covers $\lambda \in \{0, 0.1, 0.2, 0.3, 0.5, 0.7, 0.9, 0.95, 1\}$ and $\gamma \in \{0.4, 0.6, 0.8, 1, 1.25, 1.5\}$. The search maximizes block exact match and breaks ties by block NLL. Tables 11 and 12 use the inner grid $\lambda \leq 0.9$, $\gamma \leq 1$ with $k = 32$ and $\beta = 0$. The grid includes the endpoint $\lambda = 1$, where the fused distribution is the smoothed vote distribution r^ϵ itself, but this setting is never selected. With $\lambda = 0.1$ and 0.2 in the grid, the likelihood-constrained arithmetic mixture selects $\lambda = 0$ on every natural corpus (MixN in Table 7). Its EM there equals the base EM of Table 5. R-Joint selects $\lambda \geq 0.3$ on every corpus (Table 12 and Appendix H).

Block exact match is scored with the gate. Block NLL is scored under the fused distribution $q^\star$ at the selected setting, without the gate, and thus measures the likelihood of the fused expert. Table 7 checks both conventions against the ungated MAP of $q^\star$ and the deployed kernel.

Setting $Q_i = p_\theta(\cdot \mid x_t, i)$ in the proof of Proposition 1 below gives the divergence of the model's own proposal, $D_{\mathrm{KL}}\big(P \,\big\|\, p_\theta^{\otimes}\big) = \mathrm{TC}(S \mid x_t) + \sum_{i \in S} D_{\mathrm{KL}}(P_i \,\|\, p_\theta(\cdot \mid x_t, i))$. Any method that changes only per-position marginals, including token-wise kNN interpolation, can at best remove the second term. The first remains unchanged. This statement concerns a single conditioning state and simultaneous commitment. Section 4.5 tests joint commitment in a decoder that commits in several steps. Closing the gap requires joint information about y_S. Retrieved clean spans $s = (s_1, \ldots, s_b)$ supply an empirical prior over such tuples.

Proposition 1. Write $Q = \prod_i Q_i$. Then $D_{\mathrm{KL}}(P \,\|\, Q) = -H(P) - \sum_y P(y) \sum_i \log Q_i(y_i) = -H(P) + \sum_i \big[H(P_i) + D_{\mathrm{KL}}(P_i \,\|\, Q_i)\big]$. This follows because the expectation of $\log Q_i(y_i)$ under P depends only on the marginal P_i. Each $D_{\mathrm{KL}}(P_i \,\|\, Q_i) \geq 0$, with equality if and only if $Q_i = P_i$. The minimum is therefore $\sum_i H(P_i) - H(P) = \mathrm{TC}$, which is zero if and only if $P = \prod_i P_i$. □

Proposition 2. The objective is $(1 - \lambda) \sum_y q \log \frac{q}{p_\theta^{\otimes}} + \lambda \sum_y q \log \frac{q}{r^\epsilon} = \sum_y q(y) \log q(y) - \sum_y q(y) \log \big[p_\theta^{\otimes}(y)^{1-\lambda} r^\epsilon(y)^\lambda\big]$. This equals $D_{\mathrm{KL}}(q \,\|\, \tilde{q}/Z) - \log Z$, where $\tilde{q} = (p_\theta^{\otimes})^{1-\lambda}(r^\epsilon)^\lambda$. Since $r^\epsilon \geq \epsilon p_\theta^{\otimes} > 0$ wherever $p_\theta^{\otimes} > 0$, $\tilde{q}$ is finite and positive on the support, and $Z < \infty$. Strict convexity of KL in its first argument then gives the unique minimizer $q^\star = \tilde{q}/Z$. □

Proposition 3. Split the sum $Z = \sum_y \tilde{q}(y)$ into $y \in \mathcal{U}$ and $y \notin \mathcal{U}$. For $y \notin \mathcal{U}$, $r(y) = 0$, so $r^\epsilon(y) = \epsilon p_\theta^{\otimes}(y)$ and $\tilde{q}(y) = \epsilon^\lambda p_\theta^{\otimes}(y)$. Summing over $y \notin \mathcal{U}$ gives $\epsilon^\lambda[1 - \sum_{u \in \mathcal{U}} p_\theta^{\otimes}(u)]$ because $p_\theta^{\otimes}$ sums to one. Each of the $|\mathcal{U}| \leq k$ retained terms requires the b per-position log-probabilities of u and the vote mass $r(u)$, for a total cost of $O(kb)$. Exact sampling follows because the remainder is $p_\theta^{\otimes}$ restricted to the complement of $\mathcal{U}$, up to the constant ϵ^λ. □

Fusion as an opinion pool. The fusion rule of Proposition 2 takes the form of a logarithmic opinion pool (Genest & Zidek, 1986; Hinton, 2002). The new component is the expert being pooled: a noise-conditioned, non-parametric joint distribution over clean spans that can be swapped or extended without training. Although the tuple space contains $|\mathcal{V}|^b$ elements, the normalizer reduces to a sum over the retrieved support.

When the fusion is closer to P than the model's product. Propositions 2 and 3 say the kernel is the closed-form barycenter and that it can be normalized exactly. Neither says it is closer to the true joint P than the model's own product. Writing Z for the normalizer of $q^\star$, the definitions give

$$D_{\mathrm{KL}}(P \,\|\, q^\star) - D_{\mathrm{KL}}\big(P \,\big\|\, p_\theta^{\otimes}\big) = \log Z - \lambda\, \mathbb{E}_P\left[\log \frac{r^\epsilon}{p_\theta^{\otimes}}\right],$$

and the right-hand side is not non-positive in general. The fusion improves on the model's product exactly when $\lambda\, \mathbb{E}_P[\log(r^\epsilon / p_\theta^{\otimes})] > \log Z$. Because both experts are normalized and $0 \leq \lambda \leq 1$,

$Z \leq 1$ and the threshold is at most zero, so this does not require the retrieval expert to carry more log-likelihood than the model's product on its own. Fusion can improve on the base model while the retrieval expert alone is worse than it. Whether the improvement condition holds depends on the datastore and the query. So "breaking the factorization barrier" should be read as giving the reverse process a representation that is not a product, not as removing the divergence from P unconditionally. The gains we report measure when that representation helps.

B POSITIONING RELATIVE TO PRIOR WORK

Table 2 places R-Joint against the two families of Section 5 along the axes that separate them, an editable memory and a joint value over the committed block. Table 3 then holds the neighbours fixed and varies only how they vote, which is the comparison the placement turns on. Table 4 gives the repository results of §4.4. Tables 5, 6 and 7 give the calibrated controls and scoring checks of §4.2.

Table 2: Positioning of R-Joint. Extra training means method-specific training beyond pretrained backbones. DCD and PoE-Bridge use an additional pretrained autoregressive model. SARDI's retrieval procedure is training-free, although its reported experiments use reasoning-fine-tuned backbones. CoDiLA (Hersche et al., 2026) hands blocks to a trained autoregressive model. Every clean span in a raw-text corpus is a sample from the joint distribution of its tokens. To our knowledge, no prior method turns such spans into a joint reverse kernel for a frozen dLLM.

Method family	Editable memory	Joint values	Base frozen	Extra training
Bi-kNN	yes	per token	yes	yes, a fusion net
SARDI / ARAM	docs in context	no	yes	no
ADJUST	no	yes	yes	yes, a sampler
VMD / LADD / tensor-train	no	yes	no	yes
DCD	no	yes	yes	no
DAPD / DepCap / TACG	no	positions only	yes	no
DEMASK	no	positions only	yes	predictor
CoDiLA	no	local blocks	yes	AR model
PoE-Bridge	no	approximately	yes	no
R-Joint (ours)	yes	yes	yes	no

Table 3: Same neighbours, different votes (§4.2). These runs use fixed $\lambda = 0.5$ with no gate. The datastores are the context-free 133-part Wikipedia build with 3.6M entries and the 160-part code build with 4.3M entries (Table 23). B multiplies per-position votes. C and D vote for whole tuples. C retrieves only from the noise bucket that matches the query, and D retrieves from the whole index. Lower NLL is better. At $\lambda = 0.9$ the same arms give Wikipedia block EM of 36.7% for arm B and 47.0% for arm D.

	A: base	B: per-token	C: joint (bucket)	D: joint (all)
Wikipedia b=2 EM	49.49	49.19	50.21	**50.26**
Wikipedia b=2 NLL	3.401	4.103	3.360	**3.339**
Code b=2 EM	70.00	69.77	70.08	**70.17**
Wikipedia b=4 NLL	12.185	13.824	11.820	**11.789**

Table 4: Repository-local memory (§4.4). Block EM (%) at $b = 2$ on repositories created after the published training cut-off of both models. A creation date bounds when a repository was published, not when its contents were written, since vendored code, templates and copied files can be older. We therefore also drop any file with an exact sha1 match, after whitespace normalization, against the code training split, and any file whose 64 min-hashes over 5-token shingles exceed a Jaccard threshold of 0.8 against it. This removes 31 of 1,104 files. Third-party code that we cannot match may remain, so prior exposure to these repositories is unlikely but not ruled out. "Same" restricts retrieval to the query's repository. "Other" restricts retrieval to other repositories.

Model	base EM	same-repo	other-repo	ΔNLL (same)	repair rate (same / other)
LLaDA-8B	74.99	76.13 (+**1.14**)	75.03 (+0.04)	−7.2%	7.9% / 1.1%
Dream-7B	65.22	68.21 (+**2.99**)	65.19 (−0.03)	−9.6%	9.1% / 1.2%

Table 5: Three calibrated controls on the context-carrying datastores at $b = 2$, reported as EM (%) / block NLL. C0 and Ccal are R-Joint with plain votes ($k = 32$, $\beta = 0$) and with calibrated k and β, the two settings of Table 1. B uses per-token voting from the same neighbours and weights, followed by per-position fusion and a product, with its own gate and calibration. Mix is the arithmetic mixture $(1 - \lambda)p_\theta^{\otimes} + \lambda r^{\epsilon}$, with its own gate and calibration. Marg commits the product of $q^{\star}$'s own marginals at exactly the commit decisions of R-Joint (Ccal). The last two columns give paired, window-clustered bootstrap 95% intervals from 2,000 resamples for the EM differences Ccal−base and Ccal−Marg, in pp. NLL is scored under each arm's fused distribution.

Model	corpus	base	B	Mix	Marg	C0	Ccal	Ccal−base	Ccal−Marg
LLaDA	Wikipedia	49.49/3.401	49.83/3.395	50.74/3.503	49.94/3.382	50.26/3.343	50.21/3.322	[+0.29, +1.16]	[−0.05, +0.60]
LLaDA	legal	47.47/3.477	48.13/3.532	49.16/3.513	48.34/3.437	48.62/3.387	49.05/3.350	[+1.16, +2.02]	[+0.46, +0.95]
LLaDA	code	70.00/1.753	70.11/1.770	70.39/1.923	70.14/1.753	70.53/1.742	70.56/1.739	[+0.29, +0.83]	[+0.31, +0.54]
LLaDA	synthetic, all	35.83/5.024	39.17/4.146	39.29/4.018	38.33/4.216	38.58/4.166	39.53/3.980	[+3.29, +4.12]	[+1.00, +1.41]
LLaDA	synthetic, entity	14.08/7.414	19.48/5.978	19.25/5.751	17.85/6.130	18.16/6.030	19.31/5.737	[+4.65, +5.82]	[+1.18, +1.79]
LLaDA	dense, all	35.83/5.024	40.29/4.088	40.36/3.832	39.43/4.025	38.98/4.035	40.71/3.676	[+4.36, +5.37]	[+1.03, +1.52]
LLaDA	dense, entity	14.08/7.414	21.24/5.875	21.04/5.384	19.65/5.828	19.12/5.811	21.74/5.257	[+6.94, +8.40]	[+1.75, +2.46]
Dream	Wikipedia	48.04/3.650	48.33/3.714	49.21/3.739	48.31/3.615	48.56/3.559	48.69/3.534	[+0.17, +1.14]	[+0.06, +0.71]
Dream	legal	48.77/3.347	49.33/3.378	50.20/3.408	49.45/3.305	49.78/3.273	50.12/3.245	[+0.94, +1.75]	[+0.46, +0.87]
Dream	synthetic, all	31.99/5.728	36.32/4.760	34.90/4.887	35.17/4.836	35.25/5.125	36.48/4.656	[+4.04, +4.94]	[+1.01, +1.65]
Dream	synthetic, entity	12.37/8.121	19.18/6.594	16.87/6.780	16.96/6.773	17.01/7.224	19.12/6.492	[+6.12, +7.37]	[+1.70, +2.60]
Dream	dense, all	31.99/5.728	37.25/4.682	36.67/4.734	36.54/4.711	35.61/4.944	38.12/4.494	[+5.60, +6.65]	[+1.28, +1.87]
Dream	dense, entity	12.37/8.121	20.81/6.476	19.44/6.548	19.19/6.577	18.60/6.915	21.88/6.239	[+8.81, +10.22]	[+2.29, +3.12]

Table 6: The calibrated controls of Table 5 at block lengths 3 and 4, on the same query windows and datastores, with every arm calibrated separately for each block length. Results are EM (%)/block NLL, with paired window-clustered 95% intervals for the EM differences Ccal−base and Ccal−Marg, in pp. Gains are generally smaller at $b = 4$. Every Ccal−base interval stays above zero except Dream Wikipedia at $b = 3$. Twelve of the eighteen Ccal−Marg intervals lie above zero. The six that cover zero are five cells at $b = 4$ and Dream Wikipedia at $b = 3$. Per-token voting beats the joint rule in both metrics on the synthetic corpus at $b = 3$, where neighbours agree on whole names. The marginal control has higher EM on Dream Wikipedia at $b = 3$ and LLaDA Wikipedia at $b = 4$, while the joint rule keeps the lower NLL on both.

Model	corpus	b	base	B	Mix	Marg	Ccal	Ccal−base	Ccal−Marg
LLaDA	Wikipedia	3	28.39/7.313	28.84/7.279	29.51/7.345	28.93/7.270	29.43/7.123	[+0.57, +1.52]	[+0.12, +0.87]
LLaDA	Wikipedia	4	15.22/12.185	15.38/12.570	15.55/12.019	15.55/12.128	15.47/11.754	[+0.10, +0.59]	[−0.43, +0.26]
LLaDA	legal	3	25.92/7.462	26.49/8.295	27.32/7.489	26.34/7.415	27.44/7.268	[+0.98, +2.03]	[+0.75, +1.44]
LLaDA	legal	4	13.09/12.492	13.44/13.865	13.20/12.344	13.32/12.425	13.82/12.101	[+0.12, +1.41]	[+0.07, +0.95]
LLaDA	code	3	57.56/3.552	57.46/3.751	57.71/3.770	57.56/3.562	57.71/3.545	[+0.12, +0.36]	[+0.01, +0.32]
LLaDA	code	4	45.84/5.793	45.90/5.859	45.84/5.793	45.81/5.796	45.96/5.678	[+0.03, +0.42]	[−0.19, +0.50]
LLaDA	synthetic	3	19.79/8.738	21.13/7.916	20.48/8.603	20.42/8.591	20.99/8.335	[+0.77, +1.59]	[+0.32, +0.83]
LLaDA	synthetic	4	9.50/12.684	9.58/11.742	9.73/12.589	9.65/12.639	9.85/12.481	[+0.12, +0.49]	[−0.04, +0.43]
LLaDA	dense	3	19.79/8.738	20.67/7.584	21.20/8.324	20.54/8.077	21.26/8.140	[+0.95, +2.00]	[+0.43, +1.04]
LLaDA	dense	4	9.50/12.684	10.04/11.955	10.00/12.579	9.73/12.344	10.31/12.350	[+0.35, +1.30]	[+0.28, +0.91]
Dream	Wikipedia	3	26.10/7.990	26.35/8.791	26.31/8.003	26.31/7.931	26.06/7.773	[−0.28, +0.48]	[−0.92, +0.43]
Dream	Wikipedia	4	12.71/13.674	12.04/15.080	12.88/13.335	12.63/13.572	13.04/13.059	[+0.13, +0.88]	[−0.26, +1.21]
Dream	legal	3	26.83/7.288	27.20/8.055	27.81/7.303	27.07/7.223	27.85/7.104	[+0.45, +1.64]	[+0.44, +1.13]
Dream	legal	4	14.32/12.215	14.86/13.442	14.75/12.033	14.59/12.149	14.94/11.808	[+0.03, +1.22]	[−0.07, +0.74]
Dream	synthetic	3	16.21/9.842	17.78/8.526	16.88/9.659	16.70/9.284	17.43/9.340	[+0.86, +1.59]	[+0.42, +1.05]
Dream	synthetic	4	6.72/13.923	6.99/14.020	6.91/13.813	7.14/13.771	7.37/13.651	[+0.31, +1.06]	[+0.08, +0.43]
Dream	dense	3	16.21/9.842	18.33/8.739	17.49/9.288	17.13/9.159	17.86/9.106	[+1.19, +2.16]	[+0.35, +1.10]
Dream	dense	4	6.72/13.923	7.30/13.422	7.22/13.710	6.91/13.892	7.41/13.558	[+0.31, +1.08]	[+0.19, +0.82]

Table 7: Scoring-convention and baseline checks on the context-carrying datastores at $b = 2$, using all blocks. Cmap is the exact match of the ungated MAP of $q^\star$ at the selected setting. Its NLL equals the reported block NLL, which is scored without the gate. Gcal compares the NLL of the deployed kernel, which uses $q^\star$ on committed blocks and $p_\theta^\otimes$ elsewhere, against the base model. CcalN and MixN are the joint rule and arithmetic mixture selected under the constraint that calibration NLL not exceed the base model's. Copy is a context-verified copy rule. It commits the top-voted tuple u when its vote share is at least κ and $p_\theta^\otimes(u) \geq \zeta\, p_\theta^\otimes(\hat{y})$, with κ and ζ calibrated. The Copy column gives paired window-clustered 95% intervals for the EM difference Copy$-$base in pp. Base EM is in Table 5.

Model	corpus	Ccal EM	Cmap EM	Gcal NLL / base	CcalN EM	MixN EM	Copy EM (Copy$-$base)
LLaDA	Wikipedia	50.21	50.29	3.354 / 3.401	50.21	49.49	49.89 ($[+0.15, +0.66]$)
LLaDA	legal	49.05	48.99	3.380 / 3.477	49.05	47.47	47.91 ($[+0.16, +0.73]$)
LLaDA	code	70.56	70.56	1.750 / 1.753	70.56	70.00	70.10 ($[-0.01, +0.22]$)
LLaDA	synthetic	39.53	39.50	3.995 / 5.024	39.53	39.29	36.93 ($[+0.83, +1.36]$)
LLaDA	dense	40.71	40.71	3.678 / 5.024	40.71	40.36	37.30 ($[+1.15, +1.76]$)
Dream	Wikipedia	48.69	48.56	3.573 / 3.650	48.69	48.04	48.18 ($[-0.22, +0.47]$)
Dream	legal	50.12	50.12	3.281 / 3.347	50.12	48.77	49.32 ($[+0.34, +0.78]$)
Dream	synthetic	36.48	36.45	4.673 / 5.728	36.48	34.90	33.02 ($[+0.78, +1.29]$)
Dream	dense	38.12	38.08	4.531 / 5.728	38.12	36.67	33.65 ($[+1.36, +1.95]$)

C DATASTORE SIZE AND MODEL FAMILY

Table 8 compares datastore sizes across the two model families, and Appendix E discusses the results.

Table 8: Datastore size and model family. Block EM gain in pp at $b = 2$, with residual repair rates in parentheses. Compact datastores use the same number of parts for both families. For legal text the in-distribution cell is the 60K-window build, shown in the compact column by size. The last column adds text from outside the in-distribution corpus. For Wikipedia, this is the decontaminated dump in WikiText surface form, with 916K windows alone and 1.2M when joined to WikiText-103. For legal text, out-of-state opinions are joined to the in-distribution opinions, giving 383K windows for LLaDA and 387K for Dream (Appendix H). Adding out-of-distribution text gives no consistent EM gain. Measured against the 133-part WikiText-103 build of Table 10, whose gain is $+1.02$, the enlarged Wikipedia store gives $+0.90$ alone and $+0.98$ joined. The LLaDA Wikipedia full cell uses the 120-part build, so it is not comparable with the enlarged cells.

Corpus	Model	compact (≈60–70K)	full (≈180–240K)	enlarged: alone / joined
Wikipedia	LLaDA-8B	$+0.70$ (2.5%)	$+0.92$ (4.5%)	$+0.90$ (3.7%) / $+0.98$ (4.9%)
Wikipedia	Dream-7B	$+0.60$ (3.7%)	$+0.85$ (4.2%)	–
Code	LLaDA-8B	$+0.05$ (2.4%)	$+0.21$ (3.0%)	–
Code	Dream-7B	$+0.01$ (2.0%)	$+0.03$ (3.6%)	–
Legal	LLaDA-8B	$+0.95$ (6.1%), 60K	–	$+1.08$ (5.8%), 383K
Legal	Dream-7B	$+1.15$ (5.6%), 60K	–	$+1.05$ (4.5%), 387K

D COVERAGE CEILING AND RETRIEVAL CURVES

Coverage ceilings rise with corpus size. Table 9 reports the fraction of test-window n-grams that occur in a prefix of the datastore corpus. Coverage rises with every decade of corpus size across the tested range. At the full corpus size of about 10^8 tokens, 2-grams reach 89% coverage on Wikipedia and 80% on code. Coverage of 4-grams reaches 40–42%.

Hit rate rises in proportion to coverage. Figure 4 plots hit rate against datastore size, from 10^5 to 2.7×10^6 entries. Hit rate is the fraction of query blocks whose gold tuple appears among the $k = 64$ retrieved values. We divide it by the share of the same tuples that the harvested datastore values cover at each size. This harvested-tuple coverage differs from the corpus coverage of Table 9. Along each curve this ratio stays within a narrow band. The legend gives it as 75% on code, 57% on Wikipedia bigrams and 46% on Wikipedia trigrams. Across the tested sizes, the key therefore finds a roughly constant fraction of covered tuples, so hit rate and harvested-tuple coverage grow at similar rates. Both coverage and the key's miss rate bound what the memory can offer.

Some block is almost always available. With $k = 32$, at least one block per window is hit in 99.1% of code windows and 94.8% of Wikipedia windows. On the residual subset, consisting of blocks that the model alone gets wrong, the 2-gram hit rate is 25.4% on Wikipedia and 27.5% on code.

Table 9 gives the coverage ceiling discussed in Section 4.1.

Table 9: Coverage ceiling: percentage of clean query n-grams that occur anywhere in the first f of the datastore corpus. Coverage rises in every row.

Corpus	n	$f = 1\%$	10%	31.6%	100%
Wikipedia	2	40.7	71.3	81.9	**89.0**
Wikipedia	3	17.7	40.2	53.1	**65.2**
Wikipedia	4	8.5	21.2	30.7	**41.9**
Code	2	23.1	55.0	68.7	**79.6**
Code	3	11.3	30.7	41.9	**53.7**
Code	4	7.0	20.6	30.1	**40.0**

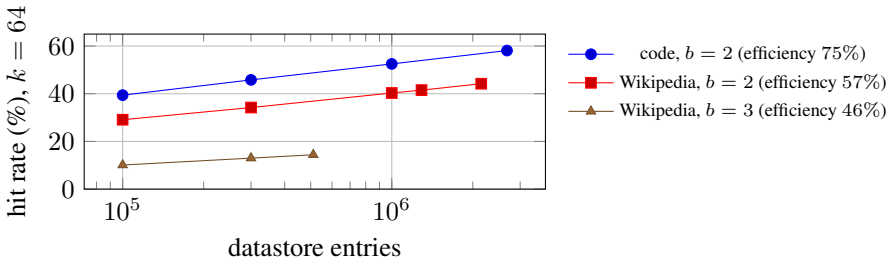

Figure 4: Retrieval hit rate versus datastore size (§4.1), with $k = 64$. The legend gives the ratio of hit rate to the coverage of the same query tuples by the datastore values at each size. This ratio stays within a narrow band along each curve, so hit rate scales with coverage. Table 9 reports coverage by corpus prefix on a coarser grid than the entry counts here.

E FURTHER BLOCK-LEVEL RESULTS

Table 10: Block EM (%) at $b = 2$ by span type, base → R-Joint (Δ pp), using plain votes with $k = 32$ on the context-free builds (Table 23). The base values in the all column match Table 1 for LLaDA and Table 5 for Dream Wikipedia and legal text, because the queries and models are the same. The repository row uses the 263-repository set, not the post-cut-off set of Table 4. In num, a gold token contains a digit. In cap, a gold token starts with a capital letter. The num and cap groups overlap. Plain contains neither type, and rare means that the rarest gold token occurs fewer than ten times in the datastore. In the order num, cap, plain and rare, the groups hold 596, 1,489, 3,935 and 69 blocks on Wikipedia, 2,450, 1,831, 11,376 and 258 on code, and 767, 2,030, 9,750 and 2,526 on the repository set.

Model	corpus	all	num	cap	plain	rare
LLaDA	Wikipedia	49.5→50.5 (+1.0)	54.7→57.4 (+**2.7**)	52.5→53.9 (+1.4)	47.5→48.1 (+0.6)	37.7→39.1 (+1.4)
LLaDA	code	70.0→70.3 (+0.3)	44.9→45.1 (+0.3)	66.1→66.5 (+0.4)	76.1→76.3 (+0.3)	47.7→48.8 (+**1.2**)
LLaDA	legal	47.5→48.4 (+1.0)	47.6→47.7 (+0.1)	47.8→50.1 (+**2.3**)	47.1→47.8 (+0.8)	33.3→34.6 (+1.3)
Dream	Wikipedia	48.0→48.9 (+0.9)	52.5→52.2 (−0.3)	50.7→51.0 (+0.3)	46.4→47.6 (+1.2)	40.6→42.0 (+1.4)
Dream	legal	48.8→49.9 (+1.2)	47.2→48.1 (+0.9)	50.6→52.6 (+**2.0**)	48.4→49.4 (+1.0)	33.1→33.6 (+0.5)
Dream	repository (same)	72.7→74.6 (+1.9)	72.5→72.6 (+0.1)	63.3→66.4 (+**3.2**)	74.6→76.4 (+1.8)	59.7→60.4 (+0.7)

Datastore mismatch. We query LLaDA's Wikipedia blocks against the 160-part code build and its code blocks against the 133-part Wikipedia build (Table 23). When calibrated on the mismatched pair, the gate closes on every Wikipedia block and fires on 0.23% of residual code blocks without harm. If we instead freeze the gate from matched calibration and swap the datastore underneath it, exact match moves by −0.20 pp on Wikipedia, with an interval of $[-0.46, +0.07]$, and −0.01 pp on code, with an interval of $[-0.08, +0.07]$. The arithmetic mixture loses 0.2 and 0.6 pp under the same swap.

Enlarged datastores. Enlarging the datastore with text from outside the in-distribution corpus gives no consistent EM gain (Table 8). A decontaminated 916K-window Wikipedia datastore gains +0.90 pp alone and +0.98 pp when joined to WikiText-103, against +1.02 pp for the 133-part WikiText-103 build alone. The out-of-distribution text therefore costs up to 0.12 pp of exact match. A 6.4× legal datastore that adds out-of-state opinions gains +1.08 against +0.95 pp on LLaDA and +1.05 against +1.15 on Dream.

The three calibrated controls in detail. *Marginals of the joint.* §4.2 and Figure 2 report this control at $b = 2$. At that length, marginal NLL is worse than the joint rule on all thirteen cells of Table 5, including code, and the Ccal−Marg exact-match intervals exclude zero except on LLaDA Wikipedia. At $b = 3$ and 4 the joint rule has lower NLL than the marginal control on fifteen of the eighteen cells of Table 6. Twelve of the eighteen Ccal−Marg intervals there lie above zero.

Calibrated per-token voting. With its own gate, this token-wise kNN analogue trails R-Joint by 0.4 to 0.9 pp on natural text at $b = 2$ (Table 5). It also raises NLL above the base on four of the five

natural rows, all but LLaDA Wikipedia. Synthetic neighbours agree on whole names, so with the standard 8-template memory its EM comes within 0.4 pp of R-Joint in both families. Its NLL is still 0.10 to 0.24 nats worse there, and 0.62 nats worse on LLaDA's dense entity blocks.

Arithmetic mixture. With its own gate, $(1-\lambda)p_\theta^{\otimes} + \lambda r^{\epsilon}$ exceeds R-Joint's EM on Wikipedia and legal text by up to 0.5 pp, but it raises NLL above the base on every natural corpus at $b = 2$ (Table 5). On LLaDA the increase is $+0.10$ nats on Wikipedia, $+0.04$ on legal text and $+0.17$ on code. On Dream it is $+0.09$ on Wikipedia and $+0.06$ on legal text. The mixture spreads mass over wrong tuples. Under the frozen-gate datastore swap described above, its exact match falls by 0.2 pp on Wikipedia and 0.6 pp on code, while that of R-Joint moves by -0.20 and -0.01 pp. Constraining each arm's calibration NLL not to exceed the base leaves R-Joint unchanged but forces the mixture to $\lambda = 0$ on every natural corpus (Table 7).

The ungated MAP of $q^\star$ matches gated EM within 0.15 pp on every corpus and family (Table 7). The deployed kernel uses $q^\star$ on committed blocks and $p_\theta^{\otimes}$ elsewhere, and its NLL is lower than the base on every corpus and family. On LLaDA it is lower by 0.05 nats on Wikipedia, 0.10 on legal text, 0.003 on code, 1.03 on synthetic blocks and 1.35 with the denser memory. The code difference has a document-level interval that covers zero (Appendix F). A context-verified copy rule commits the top-voted tuple when vote share and model probability meet two calibrated thresholds. Measured against the base EM of Table 5, it recovers 23 to 30% of the calibrated synthetic gain and 22 to 56% of the gain on Wikipedia and legal text (Table 7).

Where the gain lives: names, numbers and coverage. The averages in Table 1 mix spans whose tokens must agree, such as a name, a number, or an identifier, with runs of function words. Table 10 divides the plain-vote runs on the context-free builds and the 263-repository set by the surface form of the gold span and by the frequency of its rarest token in the datastore. This division never enters the decision rule. On Wikipedia with LLaDA, on legal text and on repository code, the gain concentrates on spans that carry a name or a number. On LLaDA, Wikipedia blocks containing a digit gain $+2.7$ pp. Blocks containing a capitalized token gain $+2.3$ pp on LLaDA legal text, $+2.0$ pp on Dream legal text and $+3.2$ pp on Dream repository code. These gains are 1.7 to 2.7 times the corpus average. LLaDA code and Dream on Wikipedia show no such concentration, and on LLaDA code every cell except rare spans moves by at most 0.4 pp. Blocks of plain words gain $+0.3$ to $+1.8$ pp. Blocks whose rarest token occurs fewer than ten times in the datastore gain $+0.5$ to $+1.4$ pp on every corpus and both families. The $2{,}526$ rare repository blocks gain $+0.7$ pp, and they form the one group large enough to resolve a gain. A few stored occurrences are therefore enough for the joint vote to carry the whole span. The two largest rare gains come from the 69 rare Wikipedia blocks. Each of these gains is a movement of one block, and its bootstrap interval covers zero. The 258 rare code blocks gain $+1.2$ pp, about four times the code average of $+0.3$ pp. That gain is a movement of three blocks. Neither of these two groups is resolved on its own.

The same spans in the context. We write each block's top-voted span into a prefix as an isolated span and score blocks from the model's own marginals, with no fusion or gate. On synthetic blocks this reaches 36.7% EM, against 38.6% with plain votes and 35.8% for the base model (Table 11). It thus recovers about a third of the plain-vote gain. It also falls below the base model on every natural corpus, by 1.3, 1.3, and 0.2 pp on Wikipedia, code, and legal text. The source-window prefix of §4.5 makes more effective use of the same retrieval inside a decoder.

Second model family and datastore scaling. Table 8 in Appendix C repeats the Wikipedia and code cells on Dream-7B. The ordering code $<$ Wikipedia holds on Dream, with gains of $+0.03$ against $+0.85$ pp. Growing the Wikipedia datastore from about 60K windows to the full build adds $+0.22$ pp on LLaDA and $+0.25$ pp on Dream, consistent with the coverage scaling in §4.1. The full build has 120 parts and about 240K windows on LLaDA, and 91 parts and about 180K windows on Dream. §4.3 and Figure 3 report the cross-family ordering on the context-carrying builds of Table 5. The figure takes only its Dream code point from the full code cell of Table 8.

Datastores. Each datastore is built from its corpus's training split in parts of up to $2{,}000$ windows. The main table uses one build per corpus that also stores block contexts. Wikipedia has 126 parts with 244K windows and 4.1M entries. Code has 151 parts with 5.0M entries, legal has 38 parts with 75K windows and 1.25M entries, and synthetic has 3 parts with 4.6K windows, or 8 parts with

15K windows for the denser memory. Several appendix tables use context-free builds of the same text, which store no block context. Among them, Wikipedia has 120 or 133 parts, code has 2.7M or 4.3M entries, and legal text has 60K windows. Table 23 lists the builds and the tables that use them. Compact datastores use the first 31 parts for Wikipedia or 35 for code, corresponding to about 62K and 70K windows.

Information isolation. The datastore may contain any corpus disjoint from the test instances, the test *inputs*, and the model's own outputs. It never contains the held-out query text. By design, the synthetic experiments store other renderings of the same facts, and the article-local experiments store other paragraphs of the same article. Throughout, training text builds the datastore, validation text selects every setting, and test text is reserved for evaluation and the stated decontamination procedure.

The synthetic corpus. The *synthetic* corpus contains 12,000 invented entities, namely persons, spouses, towns, institutes and prizes. Their names are built from invented syllables, so the model cannot know them. Facts are rendered using three disjoint sets of templates for datastore, calibration, and query text. Sentence order is shuffled globally so that two sentences about the same entity almost never share a window. A repeated person occurs in 3 of the 1,200 query windows and in none of the calibration windows. Entity blocks are query blocks whose gold tuple contains an invented word.

Repository corpora. For the local-memory study of §4.4 we use two *repository* corpora of Python files. The first is the 263-repository set of public repositories, used by Tables 10, 13, 14, 17 and 18. The second is the post-cut-off set of ten repositories created after the training cut-off of both models, each with at least 50 Python files under a permissive licence. Table 4 uses it, and its caption explains why prior exposure is unlikely though not impossible.

Injection of unseen names. Template-disjoint paraphrases improve synthetic entity EM and the probability of the correct span. On LLaDA's entity blocks that probability rises $4.0\times$ with plain votes and $5.35\times$ with calibrated k and β. The first ratio follows from the NLLs of the base and plain-vote (C0) columns of Table 5, and the second is the gold-probability column of Table 1. Adjacent unmasked name fragments help select the right record among valid invented-name neighbours, but they are not required, as the edited-world test of Appendix H shows. The same appendix describes the template sets, which are disjoint by construction.

Article-local memory. Article-local memory on Wikipedia, grouped and held out as in §4.4, adds $+0.2$ to $+0.3$ pp in both families.

Table 11 reports all block lengths for LLaDA-8B, and Table 12 reports compact and full datastores for Dream-7B. The latter also lists the calibration-selected (λ, γ). Both tables use the inner grid ($\lambda \leq 0.9, \gamma \leq 1$) of Appendix A. Several Dream cells select $\gamma = 1.0$, the strictest value of that grid, while the full grid reaches $\gamma = 1.5$.

F Robustness details

Document-level clusters. Windows from the same article, opinion or file are not independent. We therefore also compute the paired intervals of §4.3 by resampling documents, assigning each window to the document containing its first token. This gives 59 Wikipedia articles with 576 windows and at most 32 windows per article, 249 legal opinions with 1,200 windows, and 226 code files with 1,500 windows. The calibrated gains of Table 1 become $+0.72$ $[+0.21, +1.24]$ on Wikipedia, $+1.58$ $[+1.17, +1.96]$ on legal text and $+0.56$ $[+0.24, +0.88]$ on code. Document resampling is coarser than the window resampling of Table 5. It widens the interval on Wikipedia and code, and on legal text the interval comes out slightly narrower. The Ccal−Marg intervals become $[-0.04, +0.55]$, $[+0.48, +0.95]$ and $[+0.30, +0.56]$, respectively. The NLL intervals of the joint rule keep their sign. For LLaDA's deployed kernel on code, the NLL difference of -0.003 nats in Table 7 has a document-level interval of $[-0.010, +0.003]$, which covers zero. The conclusions of Table 5 do not depend on the clustering level.

Two further synthetic corpora drawn with different seeds give gains of $+3.52$ and $+3.28$ pp on entity blocks with plain votes. Together with the gain of $+4.14$ for the main corpus on its context-free 8-

Table 11: LLaDA-8B with the calibrated gate at all block lengths, on the context-free builds (Table 23). Rows not marked compact use the 120-part Wikipedia, 2.7M-entry code, 60K-window legal and 8-template synthetic builds.

Corpus	b	base EM	R-Joint EM	ΔEM (pp)	ΔNLL
Wikipedia	2	49.49	50.41	$+0.92$	
Wikipedia	3	28.39	29.38	$+0.99$	-1.8% to -3.2%
Wikipedia	4	15.22	15.89	$+0.67$	
Code	2	70.00	70.21	$+0.21$	$\approx -0.6\%$
Code	3	57.56	57.76	$+0.20$	–
Code	4	45.84	46.18	$+0.34$	–
Legal	2	47.47	48.42	$+0.95$	-2.3%
Legal	3	–	–	$+0.8$	
Legal	4	–	–	$+0.7$	
Synthetic (all)	2	35.83	38.58	$+2.75$	-17%
Synthetic (all)	3	–	–	$+1.2$	
Synthetic (all)	4	–	–	$+0.2$	
Synthetic (entity)	2	14.08	18.22	$+4.14$	-19%
Wikipedia, compact	2	49.49	50.19	$+0.70$	-1.3%
Wikipedia, compact	3	28.39	28.68	$+0.29$	-1.3%
Wikipedia, compact	4	15.22	15.80	$+0.58$	-3.2%
Code, compact	2	70.00	70.05	$+0.05$	-0.2%
Code, compact	3	57.56	57.76	$+0.20$	$+0.1\%$
Code, compact	4	45.84	46.06	$+0.22$	-1.6%

Table 12: Dream-7B with the calibrated gate on the context-free builds (Table 23). The compact builds contain 60,584 Wikipedia windows and 69,138 code windows. The full builds have 91 and 95 parts and contain 180,584 and 189,138 windows, respectively.

Corpus / datastore	b	base EM	R-Joint EM	ΔEM (pp)	base NLL	R-Joint NLL	(λ, γ)
Wikipedia, compact	2	48.04	48.64	$+0.60$	3.650	3.581	(0.5, 1.0)
Wikipedia, compact	3	26.10	26.52	$+0.42$	7.990	7.848	(0.5, 0.8)
Wikipedia, compact	4	12.71	12.63	-0.08	13.674	13.206	(0.7, 1.0)
Wikipedia, full	2	48.04	48.89	$+0.85$	3.650	3.568	(0.5, 1.0)
Wikipedia, full	3	26.10	26.60	$+0.50$	7.990	7.877	(0.7, 1.0)
Wikipedia, full	4	12.71	12.46	-0.25	13.674	13.204	(0.7, 1.0)
Code, compact	2	64.01	64.02	$+0.01$	2.471	2.448	(0.3, 0.8)
Code, compact	3	48.31	48.24	-0.07	5.077	5.083	(0.7, 1.0)
Code, compact	4	36.30	36.37	$+0.07$	8.659	8.299	(0.7, 0.6)
Code, full	2	64.01	64.04	$+0.03$	2.471	2.449	(0.5, 0.8)
Code, full	3	48.31	48.57	$+0.26$	5.077	5.059	(0.7, 1.0)
Code, full	4	36.30	37.02	$+0.72$	8.659	8.271	(0.5, 0.4)

template build (Table 11), the three corpora give a mean of $+3.65$ pp with a standard deviation of 0.44. On the context-carrying datastores we also draw three mask seeds. Their plain-vote gains, as mean $\pm$ standard deviation, are $+0.81 \pm 0.14$ pp on Wikipedia, $+1.20 \pm 0.23$ on legal text, $+2.47 \pm 0.25$ on the synthetic corpus and $+3.80 \pm 0.26$ on its entity blocks. With calibrated k and β, the gains are $+0.79 \pm 0.26$, $+1.53 \pm 0.13$, $+3.52 \pm 0.16$ and $+5.15 \pm 0.20$, respectively. On Dream, the plain-vote and calibrated gains are $+0.79 \pm 0.23$ and $+0.99 \pm 0.32$ on Wikipedia, and $+0.94 \pm 0.07$ and $+1.22 \pm 0.17$ on legal text.

A datastore built with decode-aligned masks reveals the model's most confident positions instead of random ones. It stores fewer blocks per window, so we compare it with random-mask builds whose entry counts bracket its own. On Wikipedia at $b = 2$, the decode-aligned build with 733K entries gives $+0.13$ pp $[-0.07, +0.34]$ with the calibrated gate. Random-mask builds with 268K, 537K and 1.05M entries give $+0.60$ $[+0.32, +0.90]$, $+0.53$ and $+0.52$ pp, respectively. Random masks perform better per stored entry, so we use them.

More neighbours help on Wikipedia. On LLaDA's 120-part build (Table 23), the gain at $b = 2$ is $+0.45$, $+0.90$, $+0.92$ and $+1.05$ pp for $k = 8, 16, 32, 64$, while block NLL falls monotonically from 3.354 to 3.325. On code, the gain is flat at about $+0.2$ pp for $k \geq 16$. The 1024-dimensional key build on legal text has 75K windows and gives $+1.20$ and $+1.24$ pp at $k = 32$ and $k = 64$. The

256-dimensional build is the 60K-window build of Table 11 and gives $+0.95$ and $+1.30$. At $k = 64$ the 256-dimensional keys give the larger gain from fewer windows.

G END-TO-END DETAILS

The decoder restores a window of 512 tokens whose masked positions were drawn with a random ρ. With budget T, step s fixes m_s, the masked count before any commit, and retires $\lceil m_s/(T-s+1) \rceil$ positions, joint commits first and the rest in order of model confidence (Figure 5). R-Joint adds block commits once at least 55% of positions are revealed, with at most three blocks per step. Among candidates passing (5), blocks are chosen greedily without overlap by $\tilde{q}(\hat{u}) \cdot b$. Block commits count against the step budget.

Table 13: End-to-end masked-window restoration with fixed or block-calibrated gate settings. The table reports token accuracy (%) on the masked region, using LLaDA-8B unless stated. Both decoders use the same number of forward passes. The verified rows use the context weight $\beta = 1$. The repository rows use the 263-repository set (Table 23). Table 14 reports the end-to-end runs of §4.5.

Corpus	setting	base	R-Joint	Δ
Code	T=8, n=1500, fixed $(0.5, 0.4)$	81.78	81.69	-0.09
Code	T=16, n=1500, fixed $(0.5, 0.4)$	81.83	81.56	-0.27
Wikipedia	T=16, n=590, block-calibrated	67.61	67.30	-0.31
Synthetic	T=16, n=300, plain votes	57.76	58.17	$+0.41$
Synthetic	T=16, n=1000, verified β=1	57.67	58.74	$+1.07$
Synthetic, dense	T=16, n=1000, verified β=1, k=128	57.67	59.00	$+1.33$
Wikipedia, Dream-7B	T=16, n=300, $(0.5, 1.0)$	67.23	67.04	-0.19
Code, Dream-7B	T=16, n=300, $(0.5, 0.8)$	60.78	60.51	-0.27
Legal, Dream-7B	T=16, n=300, $(0.5, 0.8)$	67.33	67.02	-0.31
Repository, Dream-7B, same-repo	T=16, n=300, $(0.7, 0.8)$	87.77	88.12	$+0.35$
Repository, Dream-7B, other-repo	T=16, n=300, $(0.7, 0.8)$	87.77	87.13	-0.64

What the control shares. In Table 16 each arm runs its own closed-loop decode and feeds its commits into its later forward passes, as in Figure 5. What the three arms share is the *rule*. They use the same retrieval schedule and call sites, the same search and the same k, the same context weighting, the same candidate-generation rule over masked runs, the same gate with the same (λ, γ), and the same per-step token budget. Every arm retrieves on the same steps, a mean of 14 of 16 (Table 21), so the comparison does not confound value selection with how often the memory is consulted.

What they do not share is the *realization*. A different committed value changes the next step's hidden states. The retrieval keys, the neighbour lists, the blocks that pass the gate and the confidence ordering of the remaining positions are therefore each arm's own. The difference the table reports is the effect of the commitment rule inside a full decoder, not the effect of swapping one value at a fixed set of positions.

Wikipedia's 576 windows come from 59 articles and are not independent, so we also compute the intervals of the two Wikipedia rows by resampling articles. The point estimates are unchanged and the intervals widen to $[-0.02, +0.27]$ and $[-0.01, +0.30]$ for R-Joint$-$Marg, and to $[-0.04, +0.23]$ and $[-0.03, +0.25]$ for R-Joint$-$B. All four intervals cover zero at the article level. The synthetic rows have no document structure to cluster on, so their intervals stay at the window level. The combination-minus-prefix difference of Table 14 does not isolate the commitment rule, because those two arms also differ in retrieval frequency.

The R-Joint$-$Marg margins of Table 16 are smaller than the Ccal$-$Marg differences computed from Table 5. That table scores every sampled block, whereas the decoder commits jointly on about a fifth of the masked tokens (Table 17). On the Wikipedia rows the ratio of the two margins is about twice the commit fraction f, because the gate picks the positions where the two rules disagree most rather than a random fifth. The figures in Table 16 are whole-window differences.

Table 14: End-to-end restoration on the context-carrying datastores against a base decoder given about 40% more steps. The table reports token accuracy (%) of the base decoder at $T = 22$ and paired differences in pp, with 95% bootstrap intervals over windows, for joint commitment at $T = 16$ (R-Joint), the retrieved source windows written into the context (prefix, $T = 16$), and their combination. Mention EM is the exact recovery of entity mentions, reported in the order base / R-Joint / prefix / combination. The last column gives seconds per window in the same order. The combination beats the prefix alone on every row with a matching memory, by $+0.27$ to $+0.34$ pp on the natural corpora, and costs at most 3% more per window than the prefix alone. With memory from other repositories the difference is only $+0.07$ pp. Times are end-to-end wall-clock per window on one 96 GB NVIDIA H20 at batch size one in BF16, with the device synchronized before and after each window. They include retrieval, prefix construction and the fused kernel. Joint commitment is the cheapest arm, at about three quarters of the base decoder's cost, because it runs 16 steps against 22 and does not lengthen the sequence. Writing the source window into the context costs 36 to 38% above the base decoder, because every forward pass after the first covers twice as many positions. These models attend bidirectionally, so a prefix cannot be encoded once and reused. Table 21 splits each arm into model compute, retrieval, fusion and prefix construction. Table 18 gives the base decoder a budget that costs at least as much as the combination on the five natural-corpus rows and the same-repository row. These runs use the Ccal settings of Table 5 for both models, except the repository rows. Those use $(\lambda, \gamma) = (0.5, 0.4)$ and $k = 32$ without context weights and are evaluated on the 263-repository set rather than the post-cut-off set of Table 4. Two mask settings appear on the synthetic corpus. The ordinary *synthetic* and *dense* rows mask the window at a random ρ and nothing else, exactly like every other row. Their mention column scores the mentions that mask happens to touch, counting a mention as recovered when every one of its masked tokens is right. The two "mentions masked" rows instead mask every entity mention in full and leave the rest of the window visible. This setting is far harder, so their mention rates are much lower. Both metrics on a row come from one decode of that row's mask. Table 20 gives the counts.

Model	corpus	n	base	R-Joint	prefix	both	mention EM	s/window
LLaDA	Wikipedia	576	67.61	$+0.33$ $[+0.17, +0.51]$	$+0.51$ $[+0.34, +0.69]$	$+0.83$ $[+0.64, +1.03]$	–	2.65 / 2.01 / 3.64 / 3.72
LLaDA	legal	600	66.30	$+0.37$ $[+0.19, +0.54]$	$+0.51$ $[+0.33, +0.68]$	$+0.84$ $[+0.64, +1.02]$	–	2.63 / 1.99 / 3.63 / 3.70
LLaDA	code	600	76.94	$+0.26$ $[+0.11, +0.42]$	$+0.21$ $[+0.02, +0.38]$	$+0.48$ $[+0.30, +0.68]$	–	2.61 / 1.98 / 3.55 / 3.63
LLaDA	synthetic	1000	57.68	$+1.08$ $[+0.93, +1.22]$	$+2.87$ $[+2.69, +3.06]$	$+3.17$ $[+2.96, +3.37]$	9.07 / 10.72 / 10.24 / 11.05	2.65 / 2.00 / 3.67 / 3.73
LLaDA	dense	1000	57.68	$+1.31$ $[+1.16, +1.45]$	$+2.89$ $[+2.72, +3.07]$	$+3.42$ $[+3.23, +3.62]$	9.07 / 10.68 / 10.05 / 10.96	2.65 / 2.00 / 3.66 / 3.73
LLaDA	synthetic, mentions masked	1000	3.57	$+0.42$ $[+0.26, +0.55]$	$+0.21$ $[+0.09, +0.34]$	$+0.62$ $[+0.47, +0.76]$	2.63 / 3.65 / 3.14 / **4.14**	2.65 / 2.01 / 3.67 / 3.74
Dream	Wikipedia	576	67.11	$+0.38$ $[+0.19, +0.56]$	$+0.61$ $[+0.43, +0.80]$	$+0.95$ $[+0.71, +1.18]$	–	2.48 / 1.88 / 3.38 / 3.45
Dream	legal	600	67.78	$+0.42$ $[+0.21, +0.61]$	$+0.31$ $[+0.15, +0.48]$	$+0.64$ $[+0.41, +0.85]$	–	2.45 / 1.86 / 3.36 / 3.43
Dream	repository, same	600	85.42	$+0.95$ $[+0.75, +1.13]$	$+1.76$ $[+1.50, +2.03]$	$+2.07$ $[+1.72, +2.43]$	–	2.51 / 1.90 / 3.41 / 3.48
Dream	repository, other	600	85.42	$+0.01$ $[-0.16, +0.17]$	$+0.26$ $[+0.11, +0.42]$	$+0.33$ $[+0.09, +0.55]$	–	2.48 / 1.88 / 3.38 / 3.44
Dream	synthetic	1000	51.92	$+1.61$ $[+1.46, +1.78]$	$+3.91$ $[+3.71, +4.08]$	$+4.36$ $[+4.17, +4.58]$	10.82 / 10.71 / 11.90 / 11.46	2.45 / 1.85 / 3.36 / 3.43
Dream	dense	1000	51.92	$+1.81$ $[+1.65, +1.98]$	$+3.97$ $[+3.79, +4.14]$	$+4.66$ $[+4.45, +4.86]$	10.82 / 11.15 / 11.36 / 11.77	2.52 / 1.90 / 3.46 / 3.53
Dream	synthetic, mentions masked	1000	3.63	$+0.47$ $[+0.33, +0.62]$	$+0.19$ $[+0.08, +0.31]$	$+0.71$ $[+0.55, +0.86]$	3.76 / 4.91 / 4.22 / **5.50**	2.47 / 1.87 / 3.39 / 3.46

A full-span metric on natural text. Token accuracy can rise while whole spans stay wrong, so we also score the maximal masked runs of the LLaDA Wikipedia end-to-end runs under the setting of Table 14. A run counts only when every one of its tokens is correct. The 576 query windows contain 58,912 such runs, a mean of 102 per window and a mean length of 2.1 tokens, which is the mask budget of $\rho \in \{0.15, 0.30, 0.50, 0.70\}$ divided by that run count. This is the quantity in the runs-restored column of Table 15, measured here at the larger step budget and on all 576 windows. The base decoder restores 63.18% of the runs, against 63.06% in that table at $T = 16$ on 400 windows. Joint commitment adds $+0.41$ $[+0.09, +0.72]$ pp, the prefix $+0.58$ $[+0.25, +0.90]$, and the combination $+0.95$ $[+0.61, +1.28]$. The ordering matches the token accuracies of Table 14, and each margin is slightly larger. A run-level metric behaves this way when the token gains are concentrated rather than spread.

The source-window prefix. The prefix arm and the combination consume the same retrieval. Step 1 runs the bare window, and from that step's hidden states we issue one retrieval call over the block positions of the initial mask pattern, with the same keys, the same k and the same context weights as the joint arm. No forward pass is added to obtain it. We collect the distinct source windows behind the returned neighbours, drop repeats by document identifier and offset, drop any window at zero distance from the query, and keep the highest-weighted one. The budget is one source window of 512 tokens, matching the length of the query window, and a window is taken whole or not at all, so a prefix is empty or one window. Across the evaluated windows, 1 to 4% end with no source window (Table 22). The prefix is attended to from step 2 onward and is never refreshed from a partial decode. Its positions are never scored, never counted in accuracy, and never

Table 15: Step sweeps with 400 windows each. The table reports token accuracy, masked runs restored entirely, adjacent positions committed in the same step that are both right, and the retrieval commits of R-Joint with their precision. Token accuracy and whole-run restoration are higher at $T = 8$ than at $T = 1$ in every series. On both families the gate commits more blocks as T grows, and their precision falls. From $T = 4$ on, R-Joint is more accurate than the base decoder on all three series. At 400 windows the paired half-width implied by Table 14 is about 0.2 pp, and on Wikipedia the accuracy difference exceeds it from $T = 8$ on. The accuracy and commit columns use the gate setting of Table 14. The runs-restored and pairs entries of R-Joint use a different gate setting.

Model	corpus	T	base acc.	R-Joint acc.	runs restored (base / R-Joint)	pairs both right (base / R-Joint)	commits	precision
Dream	Wikipedia	1	65.89	65.90	60.33 / 60.35	30.3 / 29.2	603	92.5
Dream	Wikipedia	2	66.54	66.56	61.41 / 61.46	32.7 / 31.6	1,648	59.8
Dream	Wikipedia	4	67.04	67.13	62.31 / 62.32	34.6 / 33.6	3,170	51.7
Dream	Wikipedia	8	67.25	67.55	62.94 / 62.78	34.7 / 36.3	5,807	43.2
Dream	Wikipedia	16	67.17	67.49	62.91 / 62.55	34.3 / 43.7	9,180	39.0
LLaDA	Wikipedia	1	66.35	66.36	60.55 / 60.56	31.6 / 30.6	604	94.0
LLaDA	Wikipedia	2	66.78	66.79	61.46 / 61.51	34.3 / 33.0	1,655	63.1
LLaDA	Wikipedia	4	67.26	67.39	62.43 / 62.50	36.6 / 35.4	3,169	54.4
LLaDA	Wikipedia	8	67.37	67.65	62.86 / 62.89	37.1 / 38.9	5,788	45.9
LLaDA	Wikipedia	16	67.41	67.80	63.06 / 63.04	35.9 / 44.2	9,200	41.3
LLaDA	code	1	70.84	70.84	62.08 / 62.09	47.5 / 46.7	609	87.4
LLaDA	code	2	71.06	71.03	62.61 / 62.60	56.0 / 55.1	1,662	66.1
LLaDA	code	4	71.05	71.24	62.85 / 62.92	56.8 / 56.5	3,085	58.8
LLaDA	code	8	71.08	71.26	63.12 / 63.15	57.4 / 60.3	5,640	52.7
LLaDA	code	16	71.12	71.31	63.29 / 63.23	56.5 / 62.8	8,740	50.7

Table 16: What joint value selection is worth once retrieval frequency is held fixed. All three arms run the schedule of Figure 5 with the same retrieval call sites, the same k, the same candidate rule and the same step budget. They differ only in how the committed value is chosen. R-Joint takes the top tuple of $q^\star$, Marg the product of $q^\star$'s own marginals position by position, and B per-token votes from that arm's own neighbours. The base column is token accuracy and the rest are paired differences in pp. The last two columns carry 95% window bootstrap intervals. The text of this appendix gives article-level intervals for the two Wikipedia rows.

Model	corpus	base	R-Joint	Marg	B	R-Joint−Marg	R-Joint−B
LLaDA	Wikipedia	67.61	+0.33	+0.20	+0.23	+0.13 [+0.04, +0.21]	+0.10 [+0.01, +0.18]
LLaDA	synthetic	57.68	+1.08	+0.68	+0.84	+0.40 [+0.28, +0.53]	+0.24 [+0.12, +0.37]
Dream	Wikipedia	67.11	+0.38	+0.23	+0.27	+0.15 [+0.05, +0.24]	+0.11 [+0.02, +0.21]
Dream	synthetic	51.92	+1.61	+1.17	+1.32	+0.44 [+0.31, +0.58]	+0.29 [+0.16, +0.41]

counted in the revealed fraction that opens the gate or in the per-step commit budget. These models attend bidirectionally, so a prefix token's representation above the first layer depends on the window and changes as the window is decoded. The prefix is therefore not cached. Every attended position is recomputed at every step, so a prefix doubles the work of a step rather than adding a pass. Approximate block-wise caching (Wu et al., 2025) would change what the frozen model computes, so we do not use it. The prefix-only arm stops here. The combination runs the fusion and the gate of Section 3 on the same neighbour lists, so they consume the same memory. The combination also retrieves on every gate-open step, while the prefix arm retrieves once. The clean contrast for joint commitment is therefore Table 16, which holds retrieval frequency fixed.

In Table 17, the gate commits 17 to 25% of the masked tokens. On the rows with a matching memory, the committed tokens are 1.3 to 4.6 pp more accurate than the base decoder at the same positions (Table 17).

Table 13 reports end-to-end runs at the fixed setting $(\lambda, \gamma) = (0.5, 0.4)$ and at block-calibrated settings. On code with $n = 1500$ and $T = 16$, the fixed setting gives 81.83 for the base decoder and 81.56 for R-Joint. On the 400 code windows of Table 15, the base decoder changes little with the step budget. It gives 71.06% at $T = 2$ and 71.12% at $T = 16$ (Table 15). At $T = 1$, LLaDA's base decoder is 0.3 to 1.1 pp below its $T = 16$ accuracy on code and Wikipedia.

The mentions-masked rows of Table 14 are a fact-completion probe on the synthetic corpus. They mask every entity mention in full and leave the rest of the window visible. On LLaDA the base decoder recovers 2.63% of the mentions, joint commitment 3.65%, the prefix 3.14% and their combi-

Table 17: Where the end-to-end difference comes from, decomposed over both regions of the window. f is the share of masked tokens that the gate commits jointly, a is token accuracy on those positions, and q is the accuracy of the base decoder on the same positions. $\delta_{\bar{C}}$ is the same difference on the masked positions the gate does *not* commit, which both decoders reach through their own confidence ordering. Both arms decode with full feedback. A jointly committed token is visible to every later forward pass, so the uncommitted region can change as well. Every window is decomposed on its own and the columns are then window-weighted exactly as Table 14 is, so $\Delta = f(a-q) + (1-f)\delta_{\bar{C}}$ holds row by row and the last column matches the R-Joint column of Table 14 to within rounding. The uncommitted region moves the total by at most 0.07 pp in either direction and changes no sign and no ordering within either model. The gate commits about a fifth of the masked tokens and is more accurate than the base decoder on them, by a margin that falls to $+0.12$ pp when the memory comes from other repositories.

		jointly committed				rest of the window		
Model	corpus	f	a	q	$f(a-q)$	$\delta_{\bar{C}}$	$(1-f)\delta_{\bar{C}}$	Δ
LLaDA	Wikipedia	0.226	51.11	49.83	$+0.29$	$+0.05$	$+0.04$	$+0.33$
LLaDA	legal	0.246	49.38	47.96	$+0.35$	$+0.03$	$+0.02$	$+0.37$
LLaDA	code	0.218	68.90	67.57	$+0.29$	-0.04	-0.03	$+0.26$
Dream	Wikipedia	0.222	48.07	46.58	$+0.33$	$+0.06$	$+0.05$	$+0.38$
Dream	legal	0.246	51.52	49.73	$+0.44$	-0.03	-0.02	$+0.42$
Dream	repository, same	0.192	75.90	71.32	$\mathbf{+0.88}$	$+0.09$	$+0.07$	$+0.95$
Dream	repository, other	0.166	68.50	68.38	$+0.02$	-0.01	-0.01	$+0.01$

Table 18: Budget check. The combination costs about 40% more per window than the base decoder at $T = 22$, because carrying a source window in the context doubles the attended length of every forward pass after the first. We therefore also run the base decoder at $T = 31$ on the five natural-corpus rows and the same-repository row. Its cost per window is then at least that of the combination on every row below. Nine further steps move base accuracy by at most 0.04 pp, and the combination keeps a margin of $+0.44$ to $+0.93$ pp on the five natural-corpus rows and $+2.06$ pp on the same-repository row.

Model	corpus	base T=22	base T=31	both T=16	margin	s/window (22 / 31 / both)
LLaDA	Wikipedia	67.61	67.64	68.44	$+0.80$	2.65 / 3.74 / 3.72
LLaDA	legal	66.30	66.31	67.14	$+0.83$	2.63 / 3.71 / 3.70
LLaDA	code	76.94	76.98	77.42	$+0.44$	2.61 / 3.68 / 3.63
Dream	Wikipedia	67.11	67.13	68.06	$+0.93$	2.48 / 3.49 / 3.45
Dream	legal	67.78	67.81	68.42	$+0.61$	2.45 / 3.46 / 3.43
Dream	repository, same	85.42	85.43	87.49	$+2.06$	2.51 / 3.53 / 3.48

nation 4.14%. On Dream the corresponding rates are 3.76%, 4.91%, 4.22% and 5.50%. In absolute terms, the combination recovers about one fully masked mention in twenty.

H DATASTORE AND PROTOCOL CONSTANTS

Table 23 lists the datastore builds and where each is used. Windows contain 512 tokens, and noise levels $\rho \in \{0.15, 0.30, 0.50, 0.70\}$ are drawn per window. Keys are last-layer hidden states mean-pooled over the block and projected by a fixed Gaussian matrix with seed 1337 to 256 dimensions. They are stored in fp16. LLaDA-8B uses mask id 126336 and hidden size 4096, and Dream-7B uses mask id 151666 and hidden size 3584. Values are clean tuples with $b \in \{2, 3, 4\}$, and datastore parts hold up to $2{,}000$ windows each.

For LLaDA, besides the compact build, two full context-free Wikipedia builds appear in the appendix. The 120-part build has about 240K windows and 3.2M entries. It gains $+0.92$ pp at $k = 32$ and is used for the k ablation and Tables 8 and 11. The 133-part build adds 13 parts to it and gains $+1.02$ pp (Table 10). It is used for Tables 3 and 10 and as the reference for the enlarged datastore. Table 1 uses the context-carrying builds described below. Among the other context-free builds, the

```
for each query window w:              # 512 tokens, rho per window
  x, prefix <- mask(w, rho), []
  for s = 1 .. T:                     # T = 22 for base, 16 otherwise
    h, p <- model([prefix; x])        # one pass, all recomputed
    m_s  <- masked(x)                 # before any commit in this step
    B_s  <- ceil(m_s / (T - s + 1))   # this step's budget, in tokens
    open <- (m_s <= 230)              # >= 55% of the 512 revealed
    need <- (arm in {rjoint,both} and open)
            or (arm in {prefix,both} and s == 1 and prefix == [])
    if need:                          # at most one call per step
      for each block B in masked runs of x, |B| in {2,3,4}:
        N[B] <- search(store, project(mean(h[B])), k)  # from this h
        r[B] <- context_weighted_votes(N[B])   # weights recomputed
    C, c_s <- {}, 0                   # c_s counts tokens, not blocks
    if arm in {rjoint, both} and open:
      q[B] <- barycenter(p[B], smooth(r[B]), lambda)  # this step's p
      C    <- <=3 disjoint B passing gate(q[B]), by q_top(B)*|B|
      C    <- longest prefix of C with sum(|B|) <= B_s
      c_s  <- sum(|B| for B in C)
      commit(x, C)                    # whole tuples, seen by later s
    if arm in {prefix, both} and s == 1:
      S      <- windows behind N, deduped, zero-distance dropped
      prefix <- top of S by vote weight, budget = 512   # 0 or 1
    fill(x, B_s - c_s)                # base decoder, confidence order
  score(x)                            # prefix positions never scored
```

Figure 5: The four arms. `base` never retrieves. `rjoint`, the R-Joint arm, retrieves and commits but builds no prefix. `prefix` retrieves once at step 1, builds the prefix from that retrieval and never commits jointly. `both`, the combination, does both from the same neighbour lists. The retrieval key is built from the current step's hidden states, so distances and context weights are recomputed at every retrieving step, and the fused distribution always uses the current step's model marginals. The step budget B_s is fixed from the masked count before any commit, and joint commits are debited from it in tokens, so every step retires exactly B_s positions whatever the arm, and the masked count follows one trajectory across all arms at a given T. The prefix-only arm retrieves once, at step 1, and no step retrieves more than once. No arm adds a forward pass, and every arm runs exactly its T passes, 22 for the base decoder and 16 for the other three. The prefix is not encoded separately, because these models attend bidirectionally and a prefix token's representation depends on the window, which changes at every step. Carrying a prefix therefore costs a longer pass, not an extra one.

code build of Tables 8 and 11 has 2.7M entries, the legal build has 60K windows, and the 8-template synthetic build has 4,608 windows.

The synthetic corpus contains 12,000 entities. In the standard memory each fact is rendered with eight datastore templates. Query text uses three other templates and calibration text uses two, and the three sets are disjoint by construction. Sentence order is shuffled globally, so a bidirectional model can seldom copy a name from another sentence about the same entity in its window. Rendering each fact with 8 or 24 datastore templates yields 4.6K or 15K windows, respectively. On the context-carrying builds with LLaDA at $b = 2$, the denser 24-template memory raises the plain-vote gain from $+2.75$ to $+3.15$ pp on all blocks and from $+4.08$ to $+5.04$ pp on entity blocks. Block NLL on all blocks falls from 4.17 to 4.04, against a base of 5.02 (Table 1 and the C0 column of Table 5).

Fact tests. The substituted memory is built from a seed-replicate corpus with the same templates and different entities. It is stored with contexts at 8 and 24 datastore templates, and the gate is frozen from calibration on the matched memory. At $b = 2$ on LLaDA, the paired differences of matched minus substituted memory are $+1.89$ $[+1.47, +2.32]$ on all blocks, $+2.72$ $[+2.07, +3.38]$ on entity blocks and $+1.75$ $[+0.76, +2.80]$ on entity blocks with a visible cue. With the dense memory, these differences are $+1.84$, $+2.91$ and $+4.25$, respectively. On Dream, they are $+2.42$ $[+1.99, +2.86]$, $+3.95$ $[+3.32, +4.59]$ and $+2.14$. With the dense memory they are $+1.64$, $+2.44$ and $+4.02$. Each of the five differences printed with an interval excludes zero. With the standard 8-template memory, the LLaDA differences are 51% of the calibrated gain of Table 1 on all blocks and 52% on entity

Table 19: One execution trace, LLaDA-8B on a Wikipedia window drawn at $\rho = 0.50$, so 256 of the 512 positions start masked. Columns are counted as follows. *masked* is m_s, the masked count at the start of the step and before any commit in it. Because joint commits are debited from the same budget, this trajectory is identical for all three arms that run $T = 16$. *attended* is the sequence length of that step's forward pass. *passes* and *retrievals* are cumulative and include the listed step. *joint commits* is a cumulative count of *blocks*, all of length two here, so the 29 blocks at step 16 are 58 tokens, or 22.7% of the 256 masked positions. This is the window's own value of f in the sense of Table 17, whose LLaDA Wikipedia mean is 0.226. The gate opens at step 3, when 224 positions remain masked, so the joint arm retrieves on 14 of its 16 steps and the combination on 15, the extra call being the one that builds the prefix at step 1.

arm	step	attended	prefix	passes	retrievals	masked	joint commits
base	1	512	0	1	0	256	–
base	11	512	0	11	0	136	–
base	22	512	0	22	0	11	–
R-Joint	1	512	0	1	0	256	0
R-Joint	3	512	0	3	1	224	0
R-Joint	8	512	0	8	6	144	11
R-Joint	16	512	0	16	14	16	29
prefix	1	512	0	1	1	256	–
prefix	2	1,024	512	2	1	240	–
prefix	16	1,024	512	16	1	16	–
both	1	512	0	1	1	256	0
both	2	1,024	512	2	1	240	0
both	16	1,024	512	16	15	16	31

Table 20: Mention recovery with paired intervals, which Table 14 reports as point estimates only. A mention counts as recovered when every one of its masked tokens is correct. The 1,000 synthetic windows of Table 14 hold 12,706 mentions. On the ordinary rows the window is masked with a random ρ as everywhere else, and the 6,406 mentions that mask touches are scored. The mentions-masked rows instead mask all 12,706 in full and leave the rest visible, so a mention there must be produced from context alone. Every rate in Table 14 and every difference below is an integer count over one of those two denominators. Intervals are paired bootstraps over windows, so they carry the within-window correlation between mentions. The two negative entries are both on Dream's ordinary synthetic row. There joint commitment does not improve mention recovery, and the prefix leads the combination. Token accuracy and mention recovery therefore need not move together.

Model	setting	comparison	Δ mention EM, pp (mentions)	interval
LLaDA	synthetic	R-Joint − base	+1.65 (106)	[+1.15, +2.16]
LLaDA	synthetic	combination − base	+1.98 (127)	[+1.43, +2.54]
LLaDA	synthetic	combination − prefix	+0.81 (52)	[+0.46, +1.17]
Dream	synthetic	R-Joint − base	−0.11 (−7)	[−0.27, +0.04]
Dream	synthetic	combination − base	+0.64 (41)	[+0.32, +0.97]
Dream	synthetic	combination − prefix	−0.44 (−28)	[−0.71, −0.16]
LLaDA	mentions masked	combination − base	+1.51 (192)	[+1.17, +1.86]
Dream	mentions masked	combination − base	+1.74 (221)	[+1.38, +2.11]

blocks. The Dream differences are 54% and 59% of the Ccal gains over base in Table 5. The substituted memory changes facts, surface forms and candidate coverage together. These shares therefore describe the gap between two memory conditions and do not decompose the gain causally. With the dense memory the same differences are a smaller share, 38% on LLaDA and 26 to 27% on Dream, because the total gain is larger.

The edited world re-draws the spouse of every entity from an independent stream and renders the datastore, calibration and query text with the new spouses. Blocks that overlap an edited name number 1,129 on LLaDA and 1,134 on Dream. They gain +3.0 and +4.8 pp from the edited memory, compared with +2.6 and +4.6 pp from the stale memory containing the old spouses. The paired differences over all edited-name blocks are +0.44 [−1.13, +1.98] and +0.18 [−1.38, +1.69], respectively, and blocks that do not overlap an edited name move by less than 0.2 pp. These blocks

Table 21: Where the per-window time goes, on Wikipedia. A reverse step is one forward pass over every attended position, and the 512 window positions are scored. These models attend bidirectionally, so a prefix token's representation depends on the window and changes with it. We therefore do not cache the prefix across steps, and an arm that carries one pays a longer pass at every step rather than one extra pass. Linear-layer work is $2P$ per attended token, with $P = 6{,}979{,}321{,}856$ for LLaDA-8B and $P = 6{,}525{,}288{,}448$ for Dream-7B computed from the published configurations. The output head adds $2dV$ per scored position and attention adds $4Ld$ per query-key pair. The per-window totals of Table 14 are measured end to end, with the device synchronized at the window boundary. The columns below are *derived*. We divide each arm's work by an effective throughput and report the resulting split. A source window is taken whole or not at all, so a window that carries a prefix attends 1,024 positions from step 2 while one that gets none attends 512. At the 97.6% carrying rate of Table 22 the window mean is 1,012, and that is the length the column below is computed at. We take 67 TFLOP/s, which is the rate the end-to-end decoder sustains and 45% of the H20 peak of 148 TFLOP/s. The model-column estimates for the base, R-Joint and prefix arms are then 2.566, 1.866 and 3.517 s on LLaDA and 2.412, 1.754 and 3.283 s on Dream. The combination shares the model column of the prefix arm. The model columns below are the measured totals less the derived retrieval, fusion and prefix entries. They run 3.4% above the estimates on LLaDA and 2.6% above them on Dream. On the base arm there is nothing to subtract, so its measured total is its model column, and over all thirteen rows of Table 14 that column runs 1.6 to 4.5% above the estimate. The residual covers work the estimate leaves out, such as normalization, rotary embedding, softmax, activations and kernel launches, and any gap between the sustained rate and the single rate used here. The estimate is therefore a performance model rather than a hardware bound. The retrieval and fusion columns are derived the same way. One call reads every key once, 2.1 GB on Wikipedia at about 3 TB/s, which takes 0.7 ms. About 3 ms of top-k and host work bring this to 3.7 ms per call. The joint arm calls on each step at which the gate may fire. With ρ drawn from $\{0.15, 0.30, 0.50, 0.70\}$ and the commit schedule of Figure 5, that is 16, 16, 14 and 10 of the 16 steps, a mean of 14 calls, or 52 ms. The prefix arm calls once. Datastore building runs in batches and sustains 91 TFLOP/s, which corresponds to the 700 windows per minute quoted in Appendix H. Columns are seconds and sum to the measured total of Table 14.

Model	arm	steps	passes	attended/step	model	retrieval	fusion	prefix
LLaDA	base	22	22	512	2.652	–	–	–
LLaDA	R-Joint	16	16	512	1.929	0.052	0.029	–
LLaDA	prefix	16	16	$512 \to 1{,}012$	3.635	0.004	–	0.004
LLaDA	both	16	16	$512 \to 1{,}012$	3.635	0.052	0.029	0.004
Dream	base	22	22	512	2.475	–	–	–
Dream	R-Joint	16	16	512	1.800	0.052	0.029	–
Dream	prefix	16	16	$512 \to 1{,}012$	3.369	0.004	–	0.004
Dream	both	16	16	$512 \to 1{,}012$	3.369	0.052	0.029	0.004

Table 22: Prefix length per corpus. The budget is one source window, 512 tokens, matching the length of the query window, and a source window is taken whole or not at all. A prefix is therefore empty or one window. It is empty when every retrieved neighbour comes from a window already dropped as a repeat or at zero distance from the query, which happens most often on code.

corpus	no source window	one	mean tokens
Wikipedia	2.4%	97.6%	500
legal	1.8%	98.2%	503
code	4.1%	95.9%	491
synthetic	0.9%	99.1%	507
repository	3.2%	96.8%	496

are dominated by cases in which part of the name stays visible, and a visible fragment identifies the record whichever spouse the memory holds. A subgroup of these blocks has the name hidden in full and no visible entity cue. It holds 136 blocks on LLaDA and 138 on Dream, and the base model scores 0.00 on it. On this subgroup the same differences are $+5.88$ $[+2.21, +10.29]$ and $+5.07$ $[+1.45, +9.42]$. These are eight and seven blocks out of 136 and 138, and both intervals exclude zero. At block level, the memory therefore supplies the value of the relation and not only

Table 23: Datastore builds and the tables that use them. Parts hold up to 2,000 windows. Entries are stored blocks over $b \in \{2, 3, 4\}$. Builds marked ctx store three clean tokens on each side of every block. The builds labelled compact or full and the 8-template synthetic builds without ctx store no block context, and we call them context-free. The ctx builds also serve Tables 6, 7, 16 and 20, the non-repository rows of Tables 14, 17 and 18, Figure 2, and Figure 3 apart from its Dream code point. Blank cells mark counts that are not reported. Two repository builds appear. One is a set of 263 public repositories, used by the stratified and end-to-end tables. The other is a set of ten repositories created after the training cut-off of both models, each with at least 50 Python files under a permissive licence, used by Table 4.

Model	corpus	build	parts	windows	entries	used in
LLaDA	Wikipedia	compact	31	62K		Tab. 8, 11
LLaDA	Wikipedia	full, 120 parts	120	≈240K	3.2M	k ablation, Tab. 8, 11
LLaDA	Wikipedia	full, 133 parts	133		3.6M	Tab. 3, 10, swaps
LLaDA	Wikipedia	ctx	126	244K	4.1M	Tab. 1, 5, mask seeds
LLaDA	Wikipedia	enlarged (dump)	462	916K		Tab. 8
LLaDA	code	compact	35	70K		Tab. 8, 11
LLaDA	code	full, 2.7M entries		≈180–200K	2.7M	Tab. 8, 11
LLaDA	code	full, 160 parts	160		4.3M	Tab. 3, 10, swap
LLaDA	code	ctx	151		5.0M	Tab. 1, 5
LLaDA	legal	full	30	60K		Tab. 8, 10, 11, projection check
LLaDA	legal	ctx	38	75K	1.25M	Tab. 1, 5, mask seeds
LLaDA	legal	1024-d keys	38	75K		projection check
LLaDA	legal	enlarged		383K		Tab. 8
LLaDA	synthetic	8 templates	3	4.6K		corpus seeds, spans in context, Tab. 11, 13 (n=300)
LLaDA	synthetic	ctx, 8 / 24 templ.	3 / 8	4.6K / 15K	77K / 252K	Tab. 1, 5, 13 (n=1000), memory substitution, mask seeds
LLaDA	synthetic	two seed replicates	3 / 3			corpus seeds, memory substitution with ctx (§4.3)
Dream	Wikipedia	compact / full	31 / 91	60K / 180K		Tab. 8, 10 (full), 12
Dream	Wikipedia	ctx	90	≈180K		Fig. 3, Tab. 5, mask seeds
Dream	code	compact / full	35 / 95	70K / 190K		Tab. 8, 12, Fig. 3 (full)
Dream	legal	full	30	60K		Tab. 8, 10
Dream	legal	ctx	38	75K		Fig. 3, Tab. 5, mask seeds
Dream	legal	enlarged	194	387K		Tab. 8
Dream	synthetic	ctx, 8 / 24 templ.	3 / 8	4.5K / 15K		Fig. 3, Tab. 5, memory substitution
Dream	repository	263 public repositories	10	18K		Tab. 10, 13, 14, 17, 18
LLaDA, Dream	repository	post-cut-off, 10 repositories	1			Tab. 4

the visible fragment of a name. A probe over all 1,194 fully masked target names measures the same edited-minus-stale difference in name EM and gives $+6.03$ $[+4.69, +7.45]$. Three EM units appear in this paper. Block EM scores a sampled block of b tokens. Name EM, used by this probe, scores a whole target name. Mention EM, used by Table 20, scores a whole entity mention in an end-to-end decode. Each table states which it reports.

The calibrated gate uses $\epsilon = 0.1$, a temperature equal to the median retrieved distance, and a search over (λ, γ) per corpus and block length on the calibration split, using the grid of Appendix A. It uses $k = 32$ unless k is selected on calibration, as in Table 1. The gate is a decision rule, not a probability calibration. Its objective is block exact match on held-out text, so a block that passes it is one the rule selects, not one carrying a calibrated posterior. At $\lambda > 0$ and $\gamma = 1$ the rule coincides with the global MAP of $q^\star$ up to ties, because the mass held outside the retrieval support enters both sides as $\epsilon^\lambda p_\theta^\otimes$. Other values of γ move the decision away from that MAP. Table 7 shows that the ungated MAP and the gated rule reach similar exact match.

Context-carrying datastores and calibrated k, β (Table 1). The context-carrying datastores store the three clean tokens on each side of every block. Wikipedia has 126 parts and 244K windows. Legal text has 38 parts and 75K windows, code has 151 parts, and the synthetic datastores have 3 and 8 parts. These builds use the same training text as the context-free builds, and the legal build covers 75K windows instead of 60K. Because the two kinds of build differ in their stored windows, window boundaries and mask draws, plain-vote gains at $k = 32$ differ between them. Wikipedia gains $+0.77$ on the context-carrying build, against $+0.92$ and $+1.02$ on the 120- and 133-part builds. Legal text gains $+1.15$, against $+0.95$ on the 60K-window build. Synthetic entity blocks gain $+4.08$, against $+4.14$ on the 8-template context-free build (Tables 1, 8, 10 and 11).

On these datastores, $k \in \{32, 64, 128\}$ and $\beta \in \{0, 0.5, 1, 2\}$ are selected on calibration together with (λ, γ) from one top-128 retrieval. The selected $(k, \beta, \lambda, \gamma)$ at $b = 2$ are $(64, 1, 0.3, 0.8)$ on code, $(128, 1, 0.3, 0.8)$ on Wikipedia, $(128, 1, 0.5, 0.8)$ on legal text, $(64, 1, 0.7, 1.5)$ on the synthetic corpus and $(128, 1, 0.7, 1.0)$ on its dense variant. With $k = 32$ and $\beta = 0$, the selected (λ, γ) are $(0.3, 1.0)$, $(0.3, 0.8)$, $(0.5, 0.8)$, $(0.7, 1.5)$ and $(0.7, 1.25)$, respectively. Dream selects

$(128, 0.5, 0.5, 0.8)$ on Wikipedia, $(64, 0.5, 0.5, 1.0)$ on legal text and $(128, 1, 0.7, 0.6)$ on the synthetic corpus and its dense variant. With $k = 32$ and $\beta = 0$, its selections are $(0.5, 1.0)$, $(0.3, 0.8)$ and $(0.5, 0.4)$. A masked or out-of-window query position is a wildcard. A datastore context position beyond the window edge is also ignored.

Dream-7B has its own context-carrying builds (Table 23). With plain votes and with calibrated k and β, respectively, its gains are $+0.52$ and $+0.65$ on Wikipedia, $+1.01$ and $+1.35$ on legal text, $+3.26$ and $+4.49$ on all synthetic blocks, and $+4.64$ and $+6.75$ pp on entity blocks. These gains are the C0 and Ccal cells of Table 5 minus the base cell, and the calibrated gains are the Dream points of Figure 3 other than code. We also test two gate variants. The first widens the inner grid to the full grid of Appendix A, with λ up to 1 and γ up to 1.5. The second calibrates a separate gate for each surface type of the top-voted candidate span, which is a number, a capitalized word or neither, as read from the candidate itself. Neither variant moves the gains by more than 0.2 pp on any corpus.

Enlarged Wikipedia datastore. The enlarged datastore holds 916K windows. That is $3.8\times$ the 244K windows of the context-carrying WikiText-103 build (Table 23). Joined to WikiText-103, it gives $4.8\times$ as many windows. It takes the first 600,000 articles of the 2023-11-01 English Wikipedia dump. Articles shorter than 400 characters are dropped, removing 48,765 articles. Every article whose title matches a WikiText-103 test or validation article is also dropped. The 122 protected titles yield 14 hits. We then drop every paragraph that shares a word-level 8-gram with the test or validation text. There are 358,667 protected 8-grams, and this step removes 19,381 paragraphs. An article is dropped when more than half of its paragraphs have been removed, eliminating 5,285 articles. This leaves 545,936 articles.

The dump text is then rewritten into the WikiText-103 surface convention, using Moses tokenization with spaced punctuation and the `@-@`, `@,@` and `@.@` markers. This gives datastore spans and query spans a shared surface form. The unigram cosine between the datastore and the validation split rises from 0.65 to 0.98. Without this step, the calibrated gate closes at $\lambda = 0$ on the enlarged datastore because retrieved spans never match the gold tokens. A memory must be stored in the surface form of the text it will complete.

Enlarged legal datastore. The New Hampshire opinions are joined by the first 90,000 CourtListener opinions of Pile of Law with at least 800 characters. An opinion is dropped if it shares a 32-word shingle with any New Hampshire query or calibration document. This removes 409 opinions and yields 383,020 windows for LLaDA, against 60K for the in-distribution legal build.

The code corpus is the deduplicated `codeparrot-clean-valid` set. During code calibration, a nearest neighbour at near-zero distance is also dropped so that a near-duplicate of the calibration window cannot tune the gate. We audit the split using sha1 of whitespace-normalized files for exact duplicates and 64 min-hashes over 5-token shingles with a Jaccard threshold of 0.8 for near-duplicates. The audit finds no exact duplicates. It finds a near-duplicate in the datastore split for 38 of the 1,842 query files, or 2.1%, and for 42 of the 1,841 calibration files. We split the queries of the calibrated code cell of Table 1 by these flags and score both parts under the same block-level micro-average. The 95 query windows drawn from near-duplicate files contribute 1,043 of the 15,632 length-2 blocks, or 6.7% of the weight. They gain $+1.53$ pp, which is sixteen blocks. The remaining 1,405 windows contribute 14,589 blocks and gain $+0.49$ pp, which is seventy-one blocks. Weighted by those block counts the two recombine to $+0.56$ pp, the calibrated code cell of Table 1. Near-duplicates therefore account for 0.07 pp of the calibrated code gain, and the clean-subset gain is $+0.49$ pp.

Numerical reproducibility. Re-running the same LLaDA Wikipedia query set on an otherwise idle GPU reproduces every base-model block decision bit for bit. The paired intervals in this paper measure sampling uncertainty over windows or documents, given the model's predictions. The seed replicates of Appendix F vary the mask draw and the corpus draw. We also measure re-execution variation on code, where the gain is smallest. Three re-executions of the calibrated code cell of Table 1, with the same inputs and configuration, give $+0.56$, $+0.51$ and $+0.58$ pp, a mean of $+0.55$ and a range of 0.07 pp. At $b = 3$ and $b = 4$ the same three re-executions give $+0.15$ to $+0.19$ and $+0.09$ to $+0.14$ pp. These ranges contain the LLaDA code gains of Table 6, $+0.15$ and $+0.12$ pp, whose paired intervals exclude zero. Every re-execution gives a positive code gain at all three block lengths.

Cost. Building a datastore runs the frozen model once per window at about 700 windows per minute on one 96 GB NVIDIA H20 GPU, in batches of 16 windows. The 244K-window Wikipedia datastore takes about six GPU-hours to build. Keys are 256-dimensional fp16 and occupy 2.1 GB for 4.1M entries. They are searched exactly by one matrix product per block length on the GPU, so the online cost grows with the datastore, the number of query blocks and the key dimension. The $O(kb)$ cost of Proposition 3 covers the normalization once the neighbours are found, not the search. Per-window totals are measured end to end at batch size one (Table 14), and the split within each total is derived. Inside the decoder, joint commitment at $T = 16$ costs about three quarters of the base decoder's wall-clock per window at $T = 22$, because it runs fewer steps and does not lengthen the sequence. Carrying a source window in the context instead costs 36 to 38% more than that base run, since these models attend bidirectionally and the prefix must be recomputed at every step. The combination costs about 40% more, at most 3% above the prefix alone. Table 21 gives the derived breakdown and a performance-model estimate from the model configurations. An approximate index would be needed beyond a few tens of millions of entries.