# DiffRetriever: Parallel Representative Tokens for Retrieval with Diffusion Language Models

Anonymous ACL submission


## Abstract

This paper shows how diffusion language models (DLMs) can be used as effective and efficient retrievers. Existing DLM-based retrievers (e.g., DiffEmbed) follow BERT-style encoding, representing each query or passage as a single mean-pooled vector. This ignores how DLMs are trained to generate responses through masked-position prediction under bidirectional attention, a capability that can provide stronger retrieval signals. We propose DiffRetriever, which uses the DLM's native masked-position prediction directly for retrieval. For each query or passage, DiffRetriever appends one or more masked positions, using the outputs as retrieval representations in a single forward pass. With one masked position, single-representation DiffRetriever already improves over DiffEmbed on the same backbones. DiffRetriever also naturally extends to multi-representation retrieval: DLMs process multiple masked positions jointly, enabling ColBERT-style fine-grained matching with little additional encoding latency. In autoregressive LLM retrievers, the same multi-representation strategy requires sequential decoding and therefore incurs much higher latency. DiffRetriever obtains the strongest aggregate effectiveness within our matched comparison, outperforming DiffEmbed, PromptReps, and RepLLaMA. Masked-position counts selected on training data transfer well across datasets, while per-query variation suggests headroom for adaptive allocation.[1]


## 1 Introduction

Diffusion language models (DLMs) generate responses through a masked-position prediction interface: they allocate masked output positions in advance and predict them jointly under bidirectional attention (Nie et al., 2025; Ye et al., 2025). Unlike autoregressive language models, which generate one token at a time, strictly left to right, DLMs can process several masked positions jointly.

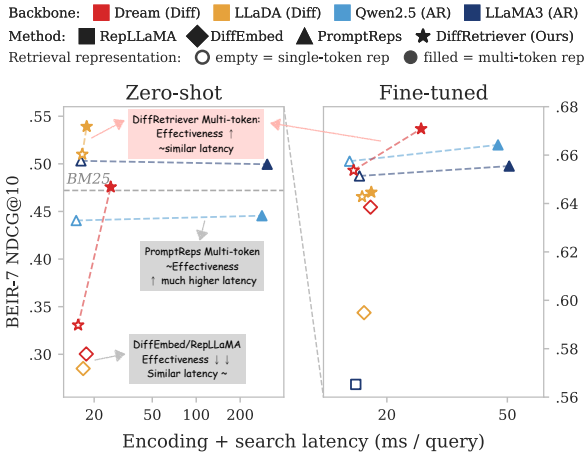


Figure 1: BEIR-7 NDCG@10 vs. query encoding plus search latency (ms/query, 100K-document sample). Left: zero-shot; right: fine-tuned. Dashed lines link single-representation and multi-representation variants. Appendix B.2 reports full latency scaling.

Existing DLM-based retrievers do not take advantage of this capability. Methods such as DiffEmbed (Zhang et al., 2025a) follow BERT-style encoding of query or passage into a single mean-pooled vector for similarity scoring. As a result, retrieval does not benefit from the masked-position prediction DLMs were trained to perform, yielding suboptimal retrieval effectiveness.

We propose a DLM retriever that directly exploits masked-position prediction for retrieval. For each query or passage, DiffRetriever prompts the model to represent the input, appends one or more masked positions after the prompt, and uses the resulting representations at those positions for retrieval. This makes DiffRetriever a natural multi-representation retriever: increasing the number of allocated masked positions increases the number of retrieval representations without requiring sequential generation.

Multi-representation retrievers were first explored with BERT-based models such as Col-

[1] Code available at https://anonymous.4open.science/r/diffretriever-75BB/.

BERT, which proved more effective than single-representation retrieval by allowing fine-grained matching between query and passage tokens (Khattab and Zaharia, 2020; Santhanam et al., 2022b). The same idea has been tested for autoregressive LLM retrievers, but did not yield consistent effectiveness gains while incurring extra cost of generating retrieval tokens sequentially (Zhuang et al., 2024). This raises a question: are multiple representations ineffective for LLM retrievers, or is sequential autoregression poorly suited to producing them? Our results point to the latter: the interface used to produce retrieval representations matters.

In both zero-shot and fine-tuned evaluation (Figure 1), even with a single masked position, DiffRetriever already improves over DiffEmbed on the same backbones, showing the value of using the masked-position prediction interface directly. Multi-representation DiffRetriever adds further gains with little additional latency, while comparable autoregressive retrievers incur much higher latency and obtain no consistent gain from multiple representations. Across MS MARCO dev, TREC DL 19/20, and BEIR-7, DiffRetriever achieves the strongest aggregate effectiveness within our matched comparison, outperforming DiffEmbed, PromptReps, and RepLLaMA. We further analyze where the gains come from by separating the effects of multiple masked-position representations and ColBERT-style MaxSim scoring. Multiple representations help even when simply mean-pooled, and MaxSim scoring adds further gains. We also study the impact of the number of masked-position tokens for representation: optimal values from training data transfer well across datasets, but per-query analysis shows significant gains by selecting an optimal number of masked tokens. Simple features such as query length and entropy correlate with this variation, suggesting adaptive masked-position prediction as a promising direction.

## 2 Related Work

**Diffusion language models in retrieval.** Diffusion language models such as Dream (Ye et al., 2025) and LLaDA (Nie et al., 2025) have motivated several retrieval methods built on DLMs, but existing methods do not use their masked-position prediction interface for retrieval. DiffEmbed (Zhang et al., 2025a) and pplx-embed (Eslami et al., 2026) treat the DLM as a BERT-style encoder, producing a single mean-pooled representation from bidirectional hidden states. DiffuRank (Liu et al., 2026) instead uses diffusion likelihood to rerank candidates from a first-stage retriever. In contrast, DiffRetriever uses the masked-position prediction interface directly, producing one or more retrieval representations in a single bidirectional forward pass.

**LLM-based and multi-representation retrieval.** Autoregressive LLM-based embedding models and retrievers such as RepLLaMA (Ma et al., 2024), E5-Mistral (Wang et al., 2024), and GTE-Qwen (Li et al., 2023) use contrastive fine-tuning to produce single-vector representations for first-stage retrieval. Recent general-purpose embedding models further improve retrieval through larger backbones and broader retrieval-oriented training data (Muennighoff et al., 2024; Lee et al., 2024; Zhang et al., 2025b). PromptReps (Zhuang et al., 2024) instead prompts autoregressive LLMs to generate a short textual representation and uses the corresponding hidden state and output logits as dense and sparse retrieval representations. PromptReps explored multi-representation variants, but found no reliable effectiveness gain over single-representation settings despite incurring sequential decoding cost. Multi-representation retrieval has been most clearly established in BERT-based late-interaction models such as ColBERT (Khattab and Zaharia, 2020; Santhanam et al., 2022b), where multiple representations per query and passage improve effectiveness through fine-grained matching. DiffRetriever revisits this idea for LLM-based retrieval through the DLM masked-position interface: multiple representations are allocated in advance and produced jointly, enabling late interaction without autoregressive sequential decoding.

## 3 Method

DiffRetriever uses masked positions in DLMs as retrieval representations. Given a query or passage, it applies a retrieval prompt and appends $K$ masked positions, where $K{=}1$ gives the single-representation setting and $K{>}1$ gives the multi-representation setting. The resulting outputs support dense, sparse, and hybrid retrieval, and the same pipeline is used for supervised contrastive fine-tuning. Figure 4 (Appendix A.1) gives an end-to-end view.

### 3.1 Retrieval Prompt

Table 1 shows the retrieval prompt. The prompt frames the task for the model in two parts: a system message that sets up an assistant role, and

Table 1: Retrieval prompt for *Query*. Square brackets indicate alternatives for single-representation ($K{=}1$) and multi-representation ($K{>}1$) cases.

| |
|---|
| *System:* You are an AI assistant that can understand human language. |
| *User:* Query: "$x$". Use [**one word** \| **a few words**] to represent the query in a retrieval task. Make sure your [**word is** \| **words are**] in lowercase.<br>*Assistant:* The [**word is** \| **words are**] " |

a user message that asks the model to produce a textual representation of a given query or passage for retrieval. The number of words requested controls the number of retrieval representations: *"one word"* for the single-representation case ($K{=}1$), and *"a few words"* for the multi-representation case ($K{>}1$).[2] The same template is used for passages, with *Query* replaced by *Passage*.

### 3.2 Parallel Masked-Position Decoding

We use the DLM masked-position prediction interface directly for retrieval. Given a retrieval prompt, we append $K$ `[MASK]` positions followed by the closing tokens of the assistant response, and pass the full sequence through the model in a single forward pass:

$$\texttt{[chat prompt]}\ \texttt{The words are "}\ \texttt{[MASK]}_1 \cdots \texttt{[MASK]}_K\ \texttt{"}\ \texttt{<turn-end>}\ \texttt{<eos>}. \tag{1}$$

The closing quote and termination tokens make the masked span a complete, length-specified assistant response. This preserves the chat format seen during DLM fine-tuning while specifying how many masked positions the diffusion backbone should predict. Here, `<turn-end>` is the chat template's end-of-turn token (e.g., `<|im_end|>` for Dream and `<|eot_id|>` for LLaDA), and `<eos>` is the end-of-sequence token.

For each `[MASK]` position, we use the hidden state and logits as retrieval signals. This yields one retrieval representation per masked position and therefore $K$ representations for the input text. Because bidirectional attention lets each `[MASK]` position attend to the prompt, the suffix tokens, and the other masked positions, the representations are produced jointly rather than sequentially. Query and passage inputs may use different numbers of masked positions; we denote these counts by $K_q$ and $K_p$.

[2]The original PromptReps multi-representation prompt used *"three words"*; we find *"a few words"* performs better. See Appendix A.3.

**Single-step decoding.** DLMs are trained as iterative denoisers: at inference, they fill masked positions over $S$ steps, revealing a subset per step and re-encoding the remaining masked positions under bidirectional attention. We report all main experiments at $S{=}1$, i.e., a single forward pass. Appendix A.2 shows that iterative denoising with $S{>}1$ is significantly worse than $S{=}1$ at zero-shot across all DLM backbones we test, and gives mixed results after fine-tuning. This suggests that the retrieval gain comes from bidirectional processing of the appended masked positions, not from the iterative denoising procedure itself.

**Comparison to PromptReps** (Zhuang et al., 2024). The same retrieval prompt can also be used with an autoregressive LLM like PromptReps. Unlike DLM decoding, where $K$ masked positions are pre-allocated before encoding, autoregressive decoding generates output tokens one-at-a-time under a causal mask until either a closing quote is produced or a generation cap $N$ is reached. Thus the actual number of generated representations varies by input and is at most $N$. The retrieval representations are otherwise analogous: each generated token provides a hidden state and logits used for retrieval. The difference is whether these representations are produced in parallel or sequentially.[3]

### 3.3 Scoring

For an input text $x$ with $K_x$ masked positions, each representation $k \in \{1, \ldots, K_x\}$ consists of a hidden state $\mathbf{h}_k \in \mathbb{R}^H$ and a logit vector $\boldsymbol{\ell}_k \in \mathbb{R}^{|V|}$. We use the hidden states to compute a dense score, the logits to compute a sparse score, and combine the two into a hybrid score by linear interpolation.

**Dense.** We score a query against a passage by matching each of the $K_q$ query representations to its closest of the $K_p$ passage representations and averaging the resulting similarities (Khattab and Zaharia, 2020):

$$s_{\text{dense}}(q, p) = \frac{1}{K_q} \sum_{i=1}^{K_q} \max_{j=1}^{K_p} \mathbf{h}_i^{q\top} \mathbf{h}_j^p, \tag{2}$$

where $\mathbf{h}_i^q$ and $\mathbf{h}_j^p$ are the $i$-th query and $j$-th passage hidden states. This naturally handles unequal $K_q$ and $K_p$ and reduces to a standard inner product when $K_q{=}K_p{=}1$.

[3]In FLOPs, diffusion has no asymptotic advantage over autoregression with KV caching: both scale linearly in the number of representations. The advantage is in wall-clock latency, since autoregressive forward passes must run sequentially while diffusion finishes in one parallel pass.

**Sparse.** For each logit vector $\boldsymbol{\ell}_k$, we apply $\log(1+\mathrm{ReLU}(\cdot))$ and aggregate across the $K_x$ representations by element-wise max-pooling over the vocabulary dimension:

$$\mathbf{s}[v] = \max_{k=1}^{K_x} \log\big(1 + \mathrm{ReLU}(\boldsymbol{\ell}_k[v])\big), \quad v \in \{1, \ldots, |V|\}. \tag{3}$$

The sparse score is the inner product $s_{\text{sparse}}(q, p) = \mathbf{s}_q^\top \mathbf{s}_p$. We apply the content-word filter from PromptReps unchanged, which restricts the sparse vocabulary to lowercase content tokens, removing stopwords, punctuation, and subword fragments before scoring.

**Hybrid.** We combine dense and sparse scores by equal-weight linear interpolation after min-max normalization within each retriever's top-1000 list, following Wang et al. (2021):

$$s_{\text{hybrid}}(q, p) = \tfrac{1}{2}\,\tilde{s}_{\text{dense}}(q, p) + \tfrac{1}{2}\,\tilde{s}_{\text{sparse}}(q, p), \tag{4}$$

where $\tilde{s}$ denotes the normalized score.

### 3.4 Supervised Fine-Tuning

We use standard contrastive fine-tuning to train each backbone for retrieval. Positive query-passage pairs are encouraged to score higher than negatives under both the dense and sparse scores. The same objective is used for all backbones. For each query $q$, let $p^+$ be a positive passage and $\mathcal{P}$ a candidate pool containing $p^+$, sampled hard negatives, and in-batch passages from other queries. The dense loss is InfoNCE with temperature $\tau$:

$$\mathcal{L}_{\text{dense}} = -\log \frac{\exp\big(s_{\text{dense}}(q, p^+)/\tau\big)}{\sum_{p \in \mathcal{P}} \exp\big(s_{\text{dense}}(q, p)/\tau\big)}. \tag{5}$$

The sparse loss $\mathcal{L}_{\text{sparse}}$ is the analogous InfoNCE loss on $s_{\text{sparse}}$, without temperature scaling (equivalently, $\tau$=1). The final objective is $\mathcal{L} = \mathcal{L}_{\text{dense}} + \mathcal{L}_{\text{sparse}}$. We use the same $(K_q, K_p)$ at training and inference time, so each backbone is trained and evaluated with the same numbers of query and passage masked positions. Full training details are given in §4.5.

## 4 Experimental Setup

### 4.1 Models

We compare two autoregressive and two diffusion LLM backbones at similar parameter scale (7 to 8B). The autoregressive models are LLaMA3-8B-Instruct (Grattafiori et al., 2024) and Qwen2.5-7B-Instruct (Team, 2024); the diffusion models are Dream-v0-Instruct-7B (Ye et al., 2025) and LLaDA-8B-Instruct (Nie et al., 2025). We refer to these as LLaMA3, Qwen2.5, Dream, and LLaDA.

We pair each diffusion backbone with a similar autoregressive model that helps isolate the impact of decoding strategy alone. Dream was initialized from Qwen2.5 and then trained with masked-position denoising, so Dream and Qwen2.5 share architecture and initialization but differ after diffusion training; this gives our closest comparison between autoregressive and diffusion backbones. LLaDA is trained from scratch as a diffusion language model, without an autoregressive checkpoint. We pair it with LLaMA3, the closest autoregressive model in size and also LLaDA's direct competitor in the original paper, as a complementary comparison without shared initialization.

### 4.2 Baselines

We compare DiffRetriever against four main baselines: (1) BM25 uses the Pyserini (Lin et al., 2021) default hyperparameters and index for each dataset. (2) PromptReps (Zhuang et al., 2024) uses Qwen2.5 and LLaMA3 as directly comparable autoregressive LLM retrieval baselines. (3) DiffEmbed (Zhang et al., 2025a) uses Dream and LLaDA as encoder-style alternatives on the same DLM backbones, mean-pooling over the input sequence without prompting or masked-position prediction. (4) RepLLaMA (Ma et al., 2024) uses LLaMA3 as a contrastively fine-tuned single-vector retrieval baseline. For PromptReps, DiffEmbed, and RepLLaMA, we re-train each baseline with the same training data, optimizer, schedule, and adapter configuration as DiffRetriever, so fine-tuned comparisons are not confounded by different training recipes. Appendix C.1 reports a broader landscape comparison without matched training recipes, including earlier BERT-based retrievers (Contriever, TAS-B, ColBERT-v2) and strong recent LLM embedding models (Qwen3-Embedding, NV-Embed, GritLM, and PPLX). In this broader comparison, DiffRetriever achieves the best in-domain MS MARCO dev result among all compared approaches. On BEIR-7 transfer, DiffRetriever outperforms earlier neural retrievers but trails the strongest dedicated embedding models, which are trained on substantially more diverse retrieval data than our MS MARCO-only fine-tuning.

### 4.3 Datasets and Metrics

We evaluate on three benchmarks. (1) *MS MARCO* passage ranking (Bajaj et al., 2016) is reported on

the dev set with MRR@10. (2) *TREC DL 2019* and *TREC DL 2020* (Craswell et al., 2020a,b) are reported with NDCG@10. (3) *BEIR-7* is a seven-dataset subset of BEIR (Thakur et al., 2021) spanning diverse domains and tasks, which we use to measure out-of-domain transfer, comprising Natural Questions, HotpotQA, SciFact, TREC-COVID, FiQA, ArguAna, and Quora; we report NDCG@10. The seven datasets span open-domain QA, multi-hop QA, scientific fact verification, biomedical retrieval, financial QA, argument retrieval, and duplicate-question detection.

For latency comparisons in Figure 1, we report query encoding plus search time using the attention backend across backbones on a 100K-document sample of the MS MARCO corpus. Appendix B.2 reports latency scaling across input length and index size. Appendix B.3 reports index-storage scaling, where multi-representation retrieval increases passage-index storage roughly with $K_p$; standard ColBERTv2 compression can reduce this by roughly $6\times$ with limited effectiveness loss.

### 4.4 Selecting $K_q$ and $K_p$

DiffRetriever processes all appended `[MASK]` positions in one forward pass, so $K_q$ and $K_p$ must be chosen before encoding. For each diffusion backbone, we sweep $(K_q, K_p) \in \{1, 2, 4, 8, 16\}^2$ on the MS MARCO training set and select the pair with the highest hybrid retrieval score, allowing $K_q$ and $K_p$ to differ. This gives $(K_q^*, K_p^*) = (4, 16)$ for Dream and $(4, 4)$ for LLaDA. Each backbone then uses its selected $(K_q, K_p)$ unchanged across all evaluations (MS MARCO dev, TREC DL 2019/2020, and BEIR-7) and as the train-time setting for supervised fine-tuning. The full selection grid is in Appendix D.1.

### 4.5 Fine-Tuning Setup

We fine-tune on the Tevatron MS MARCO passage augmented triples (Gao et al., 2022; Bajaj et al., 2016), following PromptReps. Each training item contains a query, one sampled positive passage, and 15 sampled hard negatives. Diffusion backbones are fine-tuned at their selected $(K_q^*, K_p^*)$, so training and inference use the same numbers of query and passage masked positions. For PromptReps, we use a generation cap of $N{=}4$ for both LLaMA3 and Qwen2.5 during fine-tuning.[4] Full fine-tuning hyperparameters are in Appendix B.1.

[4] Zero-shot uses $N{=}20$; we reduce the cap to $N{=}4$ during fine-tuning because larger caps exceed our memory budget at the global batch size used for matched fine-tuning.

## 5 Results

We report results in three settings: in-domain zero-shot retrieval in Section 5.1, in-domain fine-tuned retrieval in Section 5.2, and out-of-domain transfer to BEIR-7 in Section 5.3.

### 5.1 Zero-shot in-domain retrieval

Table 2 (zero-shot rows) reports zero-shot effectiveness. In the single-representation setting, PromptReps achieves the highest effectiveness: LLaMA3 reaches .242 hybrid MRR@10 on MS MARCO dev, ahead of all DiffRetriever backbones. Dream-based DiffRetriever is weakest in the single-representation setting and falls below BM25 on MS MARCO dev. DiffEmbed, which uses the same DLM backbones as mean-pooled encoders without masked-position prediction, performs even worse, indicating that masked-position representations are still preferable to encoder-style pooling even when the underlying zero-shot DLM backbone is weak.

This pattern changes in the multi-representation setting. For PromptReps, generating multiple representations yields no consistent improvement: LLaMA3 hybrid drops on MS MARCO dev and DL19, Qwen2.5 shows no significant gain across all three benchmarks, and 11 of the 18 PromptReps multi-representation cells fall below the corresponding single-representation cells. For DiffRetriever, however, the multi-representation setting improves effectiveness in every dataset and scoring mode; 16 of the 18 DiffRetriever multi-representation cells are significantly better than the corresponding single-representation cells. A scoring decomposition in Section 6.1 further shows that the gain comes from both the representations and the scoring function: mean-pooling the same multiple masked-position representations already improves over the single-representation setting, and MaxSim late-interaction scoring adds further gains.

To partially control for backbone differences, we compare Dream-based DiffRetriever with Qwen2.5-based PromptReps, since Dream was initialized from Qwen2.5 before diffusion training. Their ordering changes with the number of representations: Qwen2.5 leads in the single-representation setting, whereas multi-representation Dream leads in all cases except sparse scoring on MS MARCO dev. This suggests that the gains are tied to the parallel

Table 2: In-domain retrieval on MS MARCO dev and TREC DL. D/S/H are dense, sparse, and hybrid scores; BM25 is sparse. Fwd is query-side forward steps. **Bold**: best within each block/column. †: better than same-backbone single-rep; ‡: DiffRetriever multi-rep better than corresponding PromptReps single-rep. Paired $t$-test, $p < 0.05$.

| | Method | Backbone | Variant | Fwd | MS MARCO dev | | | DL19 | | | DL20 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | D | S | H | D | S | H | D | S | H |
| Zero-shot | BM25 | — | sparse | — | — | .184 | — | — | **.506** | — | — | **.480** | — |
| | DiffEmbed | Dream | single-rep | 1 | .034 | — | — | .184 | — | — | .110 | — | — |
| | | LLaDA | single-rep | 1 | .014 | — | — | .111 | — | — | .063 | — | — |
| | PromptReps | Qwen2.5 | single-rep | 1 | .100 | .190 | .200 | .320 | .407 | .439 | .245 | .428 | .422 |
| | | | multi-rep | ≤20 | .142† | .177 | .200 | .404 | .405 | .477 | .386† | .410 | .478† |
| | | LLaMA3 | single-rep | 1 | .178 | .211 | .242 | **.511** | .431 | .513 | .445 | .452 | **.536** |
| | | | multi-rep | ≤20 | .151 | .196 | .220 | .319 | .454 | .473 | .336 | .412 | .471 |
| | **DiffRetriever** | Dream | single-rep | 1 | .086 | .073 | .112 | .325 | .313 | .369 | .255 | .208 | .312 |
| | | | multi-rep | 1 | **.192**†‡ | .165† | .218†‡ | .479†‡ | .427† | .496† | .466†‡ | .422† | .512†‡ |
| | | LLaDA | single-rep | 1 | .158 | .200 | .221 | .391 | .451 | .481 | .367 | .463 | .483 |
| | | | multi-rep | 1 | **.192**†‡ | **.223**†‡ | **.248**† | .490† | .479 | **.549**† | **.472**† | .479 | **.536**† |
| Fine-tuned | RepLLaMA | LLaMA3 | single-rep | 1 | .412 | — | — | .715 | — | — | .690 | — | — |
| | DiffEmbed | Dream | single-rep | 1 | .405 | — | — | .720 | — | — | .693 | — | — |
| | | LLaDA | single-rep | 1 | .398 | — | — | .695 | — | — | .676 | — | — |
| | PromptReps | Qwen2.5 | single-rep | 1 | .419 | .343 | .405 | .741 | .592 | .711 | .715 | .620 | .703 |
| | | | multi-rep | ≤4 | .424 | .347 | .406 | .738 | .607 | .729 | .731 | **.632** | .705 |
| | | LLaMA3 | single-rep | 1 | .425 | .347 | .410 | .743 | .605 | .715 | **.751** | .631 | .707 |
| | | | multi-rep | ≤4 | .430 | .348 | **.414** | .746 | .613 | .732 | .739 | .621 | **.709** |
| | **DiffRetriever** | Dream | single-rep | 1 | .424 | .341 | .405 | .741 | .614 | .724 | .721 | .627 | .690 |
| | | | multi-rep | 1 | **.433**†‡ | **.349**†‡ | .411†‡ | **.751** | .620 | **.739** | .729 | .617 | .697 |
| | | LLaDA | single-rep | 1 | .424 | .347 | .405 | .715 | .621 | .704 | .715 | .624 | .701 |
| | | | multi-rep | 1 | .427 | .348 | .408 | .718 | **.636** | .718 | .721 | .614 | .698 |

masked-position interface rather than to a diffusion-backbone advantage alone.

Latency makes the tradeoff sharper. In zero-shot evaluation, PromptReps with multiple generated representations is capped at $N = 20$ and takes 275–300 ms per query, whereas DiffRetriever takes 16–20 ms (Figure 1). The roughly $15\times$ latency gap, combined with the lack of consistent multi-representation gains in PromptReps, suggests that the bottleneck is sequential generation rather than multi-representation retrieval itself.

### 5.2 In-domain fine-tuning

After contrastive fine-tuning, multi-representation DiffRetriever with Dream is the strongest system within our matched comparison (Table 2, fine-tuned rows), although margins are smaller than in the zero-shot setting. It achieves the highest score in four of the nine columns, and exceeds multi-representation PromptReps, DiffEmbed on the same Dream backbone, and RepLLaMA on MS MARCO dense retrieval. The same-backbone DiffEmbed comparison shows that, under the same training recipe, masked-position representations outperform encoder-style mean pooling.

The strongest scoring mode also shifts under supervision. Hybrid scoring is best for every in-domain zero-shot configuration, whereas dense scoring is usually strongest after fine-tuning. This suggests that contrastive supervision mainly strengthens the dense semantic signal, making equal-weight interpolation with the sparse score less helpful. We therefore report hybrid scores for zero-shot transfer and dense scores for fine-tuned transfer in the remainder of the paper.

Dream and LLaDA respond differently to the same training recipe under both DiffEmbed and DiffRetriever. In the zero-shot setting, LLaDA tends to be stronger; after fine-tuning, Dream becomes stronger. One possible explanation is the difference in initialization: Dream is initialized from Qwen2.5 and may inherit representations shaped by next-token prediction, while LLaDA is trained from scratch as a diffusion model. These inherited representations may be better suited to the discriminative matching objective used in contrastive fine-tuning. The same asymmetry appears more sharply out of domain (§5.3).

### 5.3 Out-of-domain transfer

Table 3 reports out-of-domain transfer on BEIR-7. Following the scoring-mode shift observed in §5.2, we report zero-shot results with hybrid scoring and fine-tuned results with dense scoring. The full dense/sparse/hybrid breakdown is in Appendix C.2.

**Zero-shot transfer.** The zero-shot pattern from in-domain evaluation generalizes to BEIR-7. In the single-representation setting, DiffRetriever does not improve over PromptReps: Dream trails

Table 3: Out-of-domain BEIR-7 NDCG@10. Zero-shot uses hybrid scores where available; fine-tuned uses dense scores. Fwd is query-side forward steps. **Bold**: best within each block/column. †: better than same-backbone single-rep; ‡: DiffRetriever multi-rep better than corresponding PromptReps single-rep. Paired $t$-test, $p < 0.05$.

| | Method | Backbone | Variant | Fwd | BEIR-7 (NDCG@10) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | NQ | HQA | SciFact | COVID | FiQA | ArguAna | Quora | Avg |
| Zero-shot | BM25 | — | sparse | — | .243 | **.567** | **.664** | .530 | .236 | .275 | .789 | .472 |
| | DiffEmbed | Dream | single-rep | 1 | .118 | .149 | .383 | .316 | .143 | .308 | .685 | .300 |
| | | LLaDA | single-rep | 1 | .066 | .167 | .385 | .407 | .117 | .326 | .528 | .285 |
| | PromptReps | Qwen2.5 | single-rep | 1 | .302 | .322 | .618 | .577 | .271 | .209 | .784 | .441 |
| | | | multi-rep | $\leq 20$ | .316† | .470† | .609 | .538 | .239 | .240† | .707 | .446† |
| | | LLaMA3 | single-rep | 1 | .410 | .454 | .643 | .645 | .318 | .248 | .804 | .503 |
| | | | multi-rep | $\leq 20$ | .379 | .507† | .639 | .613 | .270 | .316† | .772 | .500 |
| | **DiffRetriever** | Dream | single-rep | 1 | .153 | .066 | .380 | .497 | .220 | .273 | .725 | .331 |
| | | | multi-rep | 1 | .358†‡ | .449†‡ | .636† | .590† | .278† | .323†‡ | .696 | .476†‡ |
| | | LLaDA | single-rep | 1 | .382 | .299 | .618 | **.752** | **.325** | **.370** | **.825** | .510 |
| | | | multi-rep | 1 | **.415**† | .494†‡ | .660† | .717‡ | .317 | .357‡ | .814‡ | **.539**†‡ |
| Fine-tuned | RepLLaMA | LLaMA3 | single-rep | 1 | .622 | .503 | .653 | .564 | .383 | .411 | .821 | .565 |
| | DiffEmbed | Dream | single-rep | 1 | .625 | .668 | .735 | .758 | .478 | .375 | .831 | .638 |
| | | LLaDA | single-rep | 1 | .587 | .601 | .714 | .578 | .455 | .387 | .842 | .595 |
| | PromptReps | Qwen2.5 | single-rep | 1 | .619 | .661 | .756 | .843 | .444 | .407 | .872 | .657 |
| | | | multi-rep | $\leq 4$ | .623 | .686† | **.773**† | **.848** | .440 | .407 | .872 | .664† |
| | | LLaMA3 | single-rep | 1 | .619 | .685 | .750 | .815 | .417 | .414 | .860 | .651 |
| | | | multi-rep | $\leq 4$ | .631† | **.696**† | .735 | .809 | .437† | **.416** | .865† | .655† |
| | **DiffRetriever** | Dream | single-rep | 1 | .619 | .650 | .739 | .841 | .463 | .406 | .859 | .654 |
| | | | multi-rep | 1 | **.644**†‡ | .683†‡ | .752 | .847 | **.479**†‡ | .403 | **.887**†‡ | **.671**†‡ |
| | | LLaDA | single-rep | 1 | .620 | .640 | .733 | .840 | .453 | .414 | .799 | .643 |
| | | | multi-rep | 1 | .622 | .647† | .744 | .846 | .443‡ | .412 | .798 | .645 |

Qwen2.5, and LLaDA is roughly level with LLaMA3. With multiple representations, however, both comparisons reverse: Dream-based DiffRetriever (.476) beats Qwen2.5-based PromptReps, while LLaDA-based DiffRetriever (.539) beats LLaMA3-based PromptReps and is the strongest zero-shot BEIR-7 system in our comparison. DiffRetriever substantially outperforms DiffEmbed on the same diffusion backbones, suggesting that masked-position representations are more effective than encoder-style mean pooling for these models.

**Fine-tuned transfer.** The in-domain fine-tuned trend largely carries over to BEIR-7. Multi-representation DiffRetriever with Dream achieves the highest BEIR-7 average in our comparison (.671), ahead of multi-representation PromptReps with Qwen2.5 (.664) and DiffEmbed with Dream (.638). The improvement is statistically significant over DiffEmbed with Dream and over the single-representation LLaMA3 PromptReps baseline.

A broader pattern is that fine-tuned transfer depends strongly on the backbone family. Qwen2.5-family systems, namely DiffRetriever with Dream and PromptReps with Qwen2.5, transfer better than LLaMA-based and LLaDA-based systems. Thus, out-of-domain fine-tuned effectiveness reflects both the retrieval mechanism and the transfer behavior of the underlying backbone.

**Overall.** Across both zero-shot and fine-tuned transfer, multi-representation DiffRetriever gives the best BEIR-7 average in our comparison while retaining single-forward-pass encoding. The gains are larger with Dream than with LLaDA, showing that the benefit of the diffusion decoding interface also depends on the underlying backbone.

## 6 Analysis

The results above show that multi-representation DiffRetriever improves retrieval, but we still cannot attribute the gain to either 1) multi-token representation via a diffusion model; or 2) late-interaction MaxSim scoring (following ColBERT). To rule out the effect of late interaction, we can simply apply mean-pooling to each model.

We then turn to the impact of the number of masked tokens $(K_q, K_p)$. Recall that this was fixed based on MS MARCO train results and reused across all datasets. We investigate the impact of setting this to be dataset-specific.

### 6.1 Where do the gains come from?

Multi-representation DiffRetriever combines two changes: it uses several masked-position representations and compares them with ColBERT-style late interaction. Figure 2 separates these effects by comparing three dense scoring variants: a single representation, multiple representations with

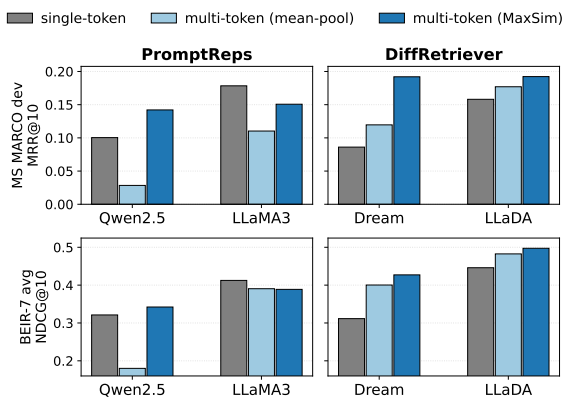


Figure 2: Dense scoring decomposition on MS MARCO dev (MRR@10) and BEIR-7 average (NDCG@10).

mean pooling, and multiple representations with ColBERT MaxSim.

For DiffRetriever, mean-pooling multiple masked-position representations already improves over the single-representation setting on both diffusion backbones. This shows that the added masked positions provide useful retrieval information even before late interaction is applied. MaxSim further adds gains, so late interaction is beneficial but not the sole source of the improvement. In contrast, PromptReps shows consistent declines from multiple generated representations under mean pooling, matching the main finding that autoregressive multi-representation retrieval pays higher decoding cost without reliable effectiveness improvements.

### 6.2 The impact of number of masked tokens

The main experiments select $(K_q, K_p)$ based on MS MARCO training and apply this to all datasets. This could be fragile if the best masked-position counts varied sharply across datasets. Figure 3 shows that this is not the case at the aggregate level. Dream selects an asymmetric, passage-heavy setting, $(K_q^*, K_p^*) = (4, 16)$, whereas LLaDA selects the symmetric setting $(4, 4)$. These train-selected cells lie at or near the best test-set cell on both MS MARCO dev and the BEIR-7 average, and the per-backbone preference persists across distributions.

However, the landscapes differ in sensitivity. On MS MARCO dev, LLaDA spans $\Delta = .07$ across grid cells, whereas Dream spans $\Delta = .14$. Thus, fixed masked-position counts remain effective in aggregate, but the cost of a poor choice is backbone-dependent and larger for Dream. [5] Appendix D.3

[5] Appendix D.2 reports results for the per-dataset optima.

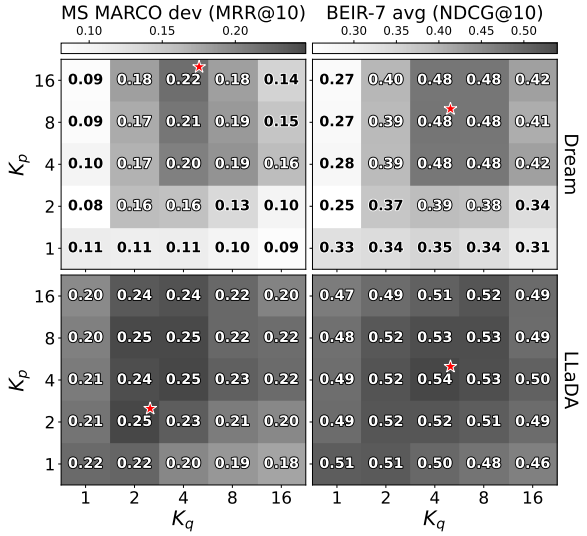


Figure 3: Zero-shot hybrid retrieval across $(K_q, K_p) \in \{1, 2, 4, 8, 16\}^2$ on MS MARCO dev and BEIR-7. Stars mark the best cell per panel; train-selected values, held fixed across evaluations, are $(4, 16)$ for Dream and $(4, 4)$ for LLaDA. Train grid: Appendix D.1.

further shows that fixed counts leave per-query headroom: on the frozen zero-shot model, an oracle that chooses $(K_q, K_p)$ separately for each query can outperform the fixed-count fine-tuned model. Although this oracle is not deployable, optimal values of $K_q$ correlate with query features cheap to compute such as length and entropy; this makes a predictor for $K_q$ and $K_p$ a promising future direction.

## 7 Conclusion

This paper shows that diffusion language models can be more effective retrievers when retrieval is formulated through their native masked-position prediction interface rather than encoder-style pooling. This interface naturally supports multi-representation retrieval by producing multiple retrieval representations in one forward pass. Even a single masked position improves over encoder-style diffusion retrieval, while multiple masked positions further improve effectiveness without the sequential generation cost that limited prior autoregressive multi-representation retrievers.

Both zero-shot and fine-tuned, DiffRetriever achieves the strongest aggregate effectiveness within our matched comparison. Our analysis also shows that different queries benefit from different numbers of masked positions, motivating adaptive masked-position allocation as a next step.

## Limitations

Our experiments focus on 7–8B-scale models, comparing two diffusion backbones (Dream and LLaDA) with two autoregressive backbones (Qwen2.5 and LLaMA3). This scale supports a controlled comparison of retrieval interfaces, especially because Dream is initialized from Qwen2.5, giving the closest available pairing between an autoregressive and diffusion backbone. Whether the interface advantage holds at other scales is an open empirical question.

Our evaluation covers MS MARCO, TREC DL 2019/2020, and a seven-dataset subset of BEIR, providing in-domain retrieval together with seven out-of-domain transfer tasks across open-domain QA, multi-hop QA, scientific, biomedical, financial, argument, and duplicate-question retrieval. We view this as sufficient to establish the central claim that the masked-position interface improves over encoder-style and autoregressive retrieval at matched scale. Extending the same interface to multilingual or long-document retrieval is a promising direction, which we leave to future work.

DiffRetriever uses fixed query and passage masked-position counts $(K_q, K_p)$, selected once on MS MARCO training data and reused across all evaluations. These counts transfer well in aggregate, but our per-query oracle analysis (App. D.3) shows substantial headroom from adaptive selection, and the correlations with cheap query features such as length and entropy suggest that a lightweight learned predictor for $(K_q, K_p)$ is a practical and promising avenue.

Multi-representation retrieval also increases passage-index storage, roughly proportional to $K_p$. This is the main systems tradeoff: DiffRetriever avoids the sequential cost of autoregressive query decoding, but still pays a storage cost on the passage side. The cost is much smaller than token-level late-interaction systems like ColBERT, and standard ColBERTv2 compression reduces it by roughly $6\times$ in our preliminary check (App. B.3) with limited effectiveness loss. Compression-aware training and learned per-passage budget allocation are natural ways to push this tradeoff further, which we leave to future work.

## Ethical considerations

We use existing public datasets, models, and software artifacts for research evaluation and comparison, including MS MARCO, TREC DL 2019/2020, BEIR, Dream, LLaDA, LLaMA3, Qwen2.5, Pyserini, PromptReps, DiffEmbed, RepLLaMA, and Tevatron. We cite the creators of these artifacts in Sections 4.1–4.5 and use them consistently with their intended research use. We do not collect or release a new dataset and do not conduct human-subject or annotation studies. AI assistants were used for grammar/typo checking, initial drafting of the paper for version 1; all AI-assisted content was reviewed, refined, and revised by the authors.

## References


Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, and 1 others. 2016. Ms marco: A human generated machine reading comprehension dataset. *arXiv preprint arXiv:1611.09268*.

Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Campos, and Ellen M Voorhees. 2020a. Overview of the TREC 2019 deep learning track. In *Proceedings of the 28th Text REtrieval Conference (TREC 2019)*. NIST.

Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Campos, and Ellen M Voorhees. 2020b. Overview of the TREC 2020 deep learning track. In *Proceedings of the Twenty-Ninth Text REtrieval Conference (TREC 2020)*. NIST.

Sedigheh Eslami, Maksim Gaiduk, Markus Krimmel, Louis Milliken, Bo Wang, and Denis Bykov. 2026. Diffusion-pretrained dense and contextual embeddings. *arXiv preprint arXiv:2602.11151*.

Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. 2022. Tevatron: An efficient and flexible toolkit for dense retrieval. *arXiv preprint arXiv:2203.05765*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Omar Khattab and Matei Zaharia. 2020. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 39–48.

Chankyu Lee, Rajarshi Roy, Mengyao Xu, Jonathan Raiman, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. 2024. Nv-embed: Improved techniques for training llms as generalist embedding models. *arXiv preprint arXiv:2405.17428*.

Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. 2023. Towards

general text embeddings with multi-stage contrastive learning. *arXiv preprint arXiv:2308.03281*.

Jimmy Lin, Xueguang Ma, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, and Rodrigo Nogueira. 2021. Pyserini: A python toolkit for reproducible information retrieval research with sparse and dense representations. In *Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval*, pages 2356–2362.

Qi Liu, Kun Ai, Jiaxin Mao, Yanzhao Zhang, Mingxin Li, Dingkun Long, Pengjun Xie, Fengbin Zhu, and Ji-Rong Wen. 2026. Diffurank: Effective document reranking with diffusion language models. *arXiv preprint arXiv:2602.12528*.

Xueguang Ma, Liang Wang, Nan Yang, Furu Wei, and Jimmy Lin. 2024. Fine-tuning llama for multi-stage text retrieval. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2421–2425.

Niklas Muennighoff, Hongjin Su, Liang Wang, Nan Yang, Furu Wei, Tao Yu, Amanpreet Singh, and Douwe Kiela. 2024. Generative representational instruction tuning. *Preprint*, arXiv:2402.09906.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. 2025. Large language diffusion models. *arXiv preprint arXiv:2502.09992*.

Keshav Santhanam, Omar Khattab, Christopher Potts, and Matei Zaharia. 2022a. Plaid: an efficient engine for late interaction retrieval. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pages 1747–1756.

Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. 2022b. ColBERTv2: Effective and efficient retrieval via lightweight late interaction. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3715–3734, Seattle, United States. Association for Computational Linguistics.

Qwen Team. 2024. Qwen2.5: A party of foundation models.

Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. Beir: A heterogenous benchmark for zero-shot evaluation of information retrieval models. *arXiv preprint arXiv:2104.08663*.

Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. 2024. Improving text embeddings with large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11897–11916.

Shuai Wang, Shengyao Zhuang, and Guido Zuccon. 2021. Bert-based dense retrievers require interpolation with bm25 for effective passage retrieval. In *Proceedings of the 2021 ACM SIGIR international conference on theory of information retrieval*, pages 317–324.

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

Siyue Zhang, Yilun Zhao, Liyuan Geng, Arman Cohan, Anh Tuan Luu, and Chen Zhao. 2025a. Diffusion vs. autoregressive language models: A text embedding perspective. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 4273–4303, Suzhou, China. Association for Computational Linguistics.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. 2025b. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*.

Shengyao Zhuang, Xueguang Ma, Bevan Koopman, Jimmy Lin, and Guido Zuccon. 2024. Promptreps: Prompting large language models to generate dense and sparse representations for zero-shot document retrieval. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 4375–4391.

# A Supplementary Method Details

This section provides supplementary method details that support the design choices in §3.

## A.1 Architecture Overview

Figure 4 gives an end-to-end view of the DiffRetriever pipeline. A query and a passage are formatted with a retrieval prompt ending in $K_q$ or $K_p$ `[MASK]` positions. A diffusion language model processes all masked positions in one bidirectional forward pass, producing hidden states for dense retrieval and next-token logits for sparse retrieval. Scoring uses ColBERT-style MaxSim on dense vectors and max-pooled logits on sparse vectors, followed by hybrid interpolation. The top-left inset contrasts this with PromptReps on an autoregressive backbone: PromptReps generates retrieval representations sequentially up to a cap $N$, whereas DiffRetriever pre-allocates $K$ masked positions and processes them in parallel.

## A.2 Multi-Step Denoising

The main paper uses a single forward pass (§3.2, $S$=1) to process all appended masked positions,

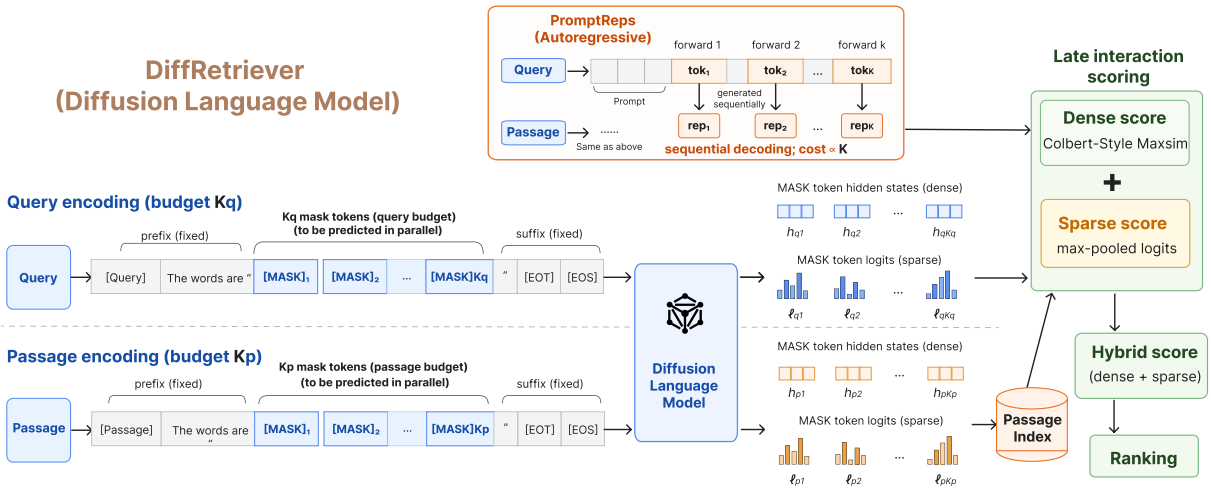


Figure 4: Overview of DiffRetriever. A query and a passage are each formatted with a retrieval prompt that ends in a sequence of $K_q$ (query) or $K_p$ (passage) `[MASK]` positions, capped by fixed suffix tokens. A diffusion language model reads all masked positions in a single bidirectional forward pass, yielding $K$ hidden states for dense retrieval and $K$ next-token logit vectors for sparse retrieval on each side. Scoring uses ColBERT-style MaxSim on the dense vectors and max-pooled logits on the sparse vectors, followed by hybrid score interpolation for ranking. Top-left inset: PromptReps on an autoregressive backbone generates retrieval representations sequentially up to a maximum budget $N$, so wall-clock encoding cost grows with the number generated; DiffRetriever pre-allocates $K$ masked positions and reads them in parallel from one pass.

since retrieval requires only the masked-position representations rather than committed token predictions. For completeness, this appendix also evaluates the multi-step variant ($S>1$), where the model iteratively denoises the masked sequence over $S$ steps by revealing a subset of positions at each step and re-encoding the remaining ones, at the multi-representation budgets used in the main paper.

**Procedure.** With $S$ denoising steps, the $K$ masked positions are unmasked over $S$ rounds following the confidence-based schedule of the underlying diffusion backbone (Nie et al., 2025; Ye et al., 2025), with roughly $K/S$ positions unmasked per step. At each step, the model performs one forward pass and replaces the most confident remaining `[MASK]` symbols with their argmax predicted tokens. The hidden state and logits at each newly unmasked position are stored in a frozen buffer. At the final step, representations from the buffer and the current forward pass are combined, and the dense and sparse scores from §3.3 are computed over this combined representation set.

**Fine-tuning interaction.** When multi-step decoding is applied during supervised fine-tuning (§3.4), the retrieval loss is computed only at the final step. The frozen buffer carries no gradient, while positions represented in the current final forward pass contribute gradients normally.

**Empirical effect.** We compare $S=1$ and $S=2$ for both diffusion backbones at the train-selected $(K_q^*, K_p^*)$ used in the main paper, across zero-shot and fine-tuned settings on MS MARCO, DL19, DL20, and the seven BEIR-7 datasets. Tables 4 and 5 report the full breakdown.

At zero-shot, $S=1$ wins decisively. $S=2$ is significantly worse than $S=1$ on every benchmark and every BEIR-7 dataset for both backbones, often by large margins: Dream MS MARCO hybrid drops from .218 to .172 (‡), and the BEIR-7 dense average drops from .427 to .309 for Dream and from .497 to .385 for LLaDA (both ‡). These results support the single-pass choice used in the main paper: zero-shot retrieval benefits from processing the appended masked positions directly, while iterative denoising degrades effectiveness in this setting.

After fine-tuning, the effect depends on the backbone. For Dream, $S=2$ has no consistent benefit: MS MARCO dense and BEIR-7 dense average both decrease, with only ArguAna showing a significant BEIR-7 gain. For LLaDA, $S=2$ improves more broadly, raising the BEIR-7 dense average from .645 to .662 (‡), but MS MARCO dense drops from .427 to .414. Because $S=1$ is consistently best at zero-shot, avoids backbone-specific inference choices, and preserves the single-pass latency advantage, we use $S=1$ throughout the main paper. Selective $S>1$ decoding for specific backbones or

Table 4: Multi-step denoising on in-domain benchmarks at the train-selected $(K_q^*, K_p^*)$: Dream $(4, 16)$, LLaDA $(4, 4)$. $S$=1 is the single-pass variant used in the main paper; $S$=2 is two-step iterative denoising. The $S$ column is the number of denoising passes; D/S/H denote dense, sparse, and hybrid scores. MS MARCO dev uses MRR@10; DL19 and DL20 use NDCG@10. **Bold**: better of the $S$=1/$S$=2 pair within each (backbone, regime, column). Underline: best score across all rows in the column. Paired two-sided t-tests compare $S$=2 against $S$=1: † ($p < 0.05$), ‡ ($p < 0.01$). $S$=2 does not consistently improve over $S$=1 in any configuration.

| | | | MS MARCO dev | | | DL19 | | | DL20 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Regime | $S$ | D | S | H | D | S | H | D | S | H |
| **Dream** | Zero-shot | $S$=1 | **.192** | **.165** | **.218** | **.479** | **.427** | **.496** | **.466** | **.422** | **.512** |
| | | $S$=2 | .102‡ | .132‡ | .172‡ | .287‡ | .393 | .465 | .317‡ | .373† | .446‡ |
| | Fine-tuned | $S$=1 | **.433** | **.349** | **.411** | .751 | .620 | .739 | .729 | .617 | .697 |
| | | $S$=2 | .425‡ | .344† | .409 | **.759** | **.622** | **.741** | **.746** | **.625** | **.709** |
| **LLaDA** | Zero-shot | $S$=1 | **.192** | **.223** | **.248** | **.490** | **.479** | **.549** | **.472** | **.479** | **.536** |
| | | $S$=2 | .132‡ | .208‡ | .205‡ | .392‡ | .477 | .513 | .374‡ | .464 | .483† |
| | Fine-tuned | $S$=1 | **.427** | **.348** | **.408** | .718 | **.636** | **.718** | .721 | **.614** | **.698** |
| | | $S$=2 | .414‡ | .330‡ | .397‡ | **.719** | .624 | .715 | **.724** | .594 | .690 |

domains is left to future work.

### A.3 Prompt Comparison: "A Few Words" vs. "Three Words"

PromptReps's multi-token representation variant (Zhuang et al., 2024) instructs the model to produce *three words* as a short textual representation of the input. We use *a few words* instead (Table 1), since the variable-budget setting in DiffRetriever is not tied to a fixed count of three and the more flexible phrasing aligns better with sweeping over different $K$ values. Since the original PromptReps multi-representation experiments were conducted on LLaMA3-8B, we compare both phrasings on LLaMA3-8B and Qwen2.5-7B.

**Setup.** We run zero-shot retrieval with multiple generated representations on MS MARCO dev, TREC DL19, and TREC DL20 under each phrasing, holding all other settings fixed: the same scoring functions, hybrid interpolation, and $N$=20 zero-shot cap on autoregressive sequential generation.

**Result.** Table 6 reports the comparison, and the two backbones differ. On LLaMA3-8B, *"a few words"* wins on 8 of 9 benchmark–scoring cells, with significant dense and hybrid gains across all three benchmarks. For example, dense MRR@10 on MS MARCO rises from .094 to .151, and dense NDCG@10 on DL19 rises from .216 to .319. On Qwen2.5-7B, the ranking is less uniform: *"three words"* wins 5 of 9 cells, mostly on sparse and hybrid, while dense is statistically tied on MS MARCO and only marginally different on DL19 and DL20. Thus, the two phrasings are not strictly ordered across backbones; the gap is larger for LLaMA3 than for Qwen2.5, and the win direction differs between them.

We use *"a few words"* as the default phrasing in the main paper for three reasons. First, it does not impose a fixed semantic count such as *"three"*, which is better aligned with a variable-representation setting. Second, it improves the original PromptReps backbone, LLaMA3, where the gap between phrasings is largest. Third, the same prompt is used uniformly across all backbones, so comparisons between decoding strategies are not affected by backbone-specific prompt tuning.

## B Supplementary Experimental Setup

This section gives implementation and measurement details for the experiments in §4.

### B.1 Fine-Tuning Details

Table 7 lists the fine-tuning recipe. We use the same optimization, batching, and LoRA settings across all four backbones and re-trained baselines, so fine-tuned comparisons are not confounded by different training recipes. LoRA target module names differ between LLaMA-/Qwen-style backbones and LLaDA, but we target the same functional set of weight matrices: attention $\{q, k, v, o\}$ projections and the three feed-forward projections. The trainable-parameter fraction stays within 0.52–0.53% across all four backbones (40–42 M parameters).

The autoregressive baselines train with a maximum generation budget of $N$=4, rather than the zero-shot cap $N$=20, because longer sequential generation exceeds our memory budget at the

Table 5: Multi-step denoising on BEIR-7 (out-of-domain). Per-dataset NDCG@10 plus the BEIR-7 average, at the train-selected $(K_q^*, K_p^*)$: Dream $(4, 16)$, LLaDA $(4, 4)$. $S$=1 is the single-pass variant used in the main paper; $S$=2 is two-step iterative denoising. The $S$ column is the number of denoising passes; D/S/H denote dense, sparse, and hybrid scores. **Bold**: better of the $S$=1/$S$=2 pair within each (backbone, regime, column). Underline: best score across all rows in the column. Paired two-sided t-tests compare $S$=2 against $S$=1 (pooled across the seven BEIR datasets for Avg): † ($p < 0.05$), ‡ ($p < 0.01$). $S$=2 does not consistently improve over $S$=1 in any configuration.

| | | | NQ | | | HQA | | | SciFact | | | COVID | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Regime | $S$ | D | S | H | D | S | H | D | S | H | D | S | H |
| **Dream** | Zero-shot | $S$=1 | **.346** | **.219** | **.358** | **.294** | **.411** | **.449** | **.554** | **.564** | **.636** | **.622** | **.481** | **.590** |
| | | $S$=2 | .179‡ | .173‡ | .267‡ | .121‡ | .292‡ | .301‡ | .419‡ | .461‡ | .559‡ | .382‡ | .313‡ | .495‡ |
| | Fine-tuned | $S$=1 | **.644** | **.458** | **.596** | **.683** | .603 | **.705** | .752 | **.666** | **.729** | **.847** | **.665** | **.830** |
| | | $S$=2 | .643 | .455 | .596 | .662‡ | **.608**‡ | .697‡ | **.753** | .666 | .723 | .825 | .658 | .789‡ |
| **LLaDA** | Zero-shot | $S$=1 | **.385** | **.292** | **.415** | **.328** | **.468** | **.494** | **.592** | **.624** | **.660** | **.662** | **.690** | **.717** |
| | | $S$=2 | .255‡ | .267‡ | .328‡ | .177‡ | .424‡ | .385‡ | .463‡ | .605 | .607‡ | .571‡ | .638‡ | .697 |
| | Fine-tuned | $S$=1 | .622 | **.452** | **.584** | .647 | .613 | .687 | .744 | **.695** | .746 | **.846** | **.710** | **.819** |
| | | $S$=2 | **.634**‡ | .429‡ | .582 | **.660**‡ | **.615** | **.696**‡ | **.761** | .676 | **.747** | .841 | .659‡ | .808 |

| | | | FiQA | | | ArguAna | | | Quora | | | Avg | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Regime | $S$ | D | S | H | D | S | H | D | S | H | D | S | H |
| **Dream** | Zero-shot | $S$=1 | **.293** | **.173** | **.278** | **.350** | **.137** | **.323** | .532 | **.647** | .696 | **.427** | **.376** | **.476** |
| | | $S$=2 | .179‡ | .127‡ | .219‡ | .229‡ | .080‡ | .196‡ | **.654**‡ | .517‡ | **.707**‡ | .309‡ | .280‡ | .392‡ |
| | Fine-tuned | $S$=1 | .479 | **.316** | **.431** | .403 | **.305** | .382 | **.887** | **.748** | **.859** | **.671** | **.537** | **.647** |
| | | $S$=2 | **.485** | .313 | .429 | **.417**‡ | .295‡ | **.396**‡ | .878‡ | .747 | .854‡ | .666‡ | .535 | .640‡ |
| **LLaDA** | Zero-shot | $S$=1 | **.308** | **.225** | **.317** | **.386** | **.179** | **.357** | **.819** | **.678** | **.814** | **.497** | **.451** | **.539** |
| | | $S$=2 | .224‡ | .215† | .279‡ | .324‡ | .173† | .309‡ | .680‡ | .663‡ | .754‡ | .385‡ | .427‡ | .480‡ |
| | Fine-tuned | $S$=1 | .443 | **.303** | .397 | .412 | .291 | .389 | .798 | .595 | .760 | .645 | **.523** | .626 |
| | | $S$=2 | **.465**‡ | .294 | **.412**‡ | **.428**‡ | **.303**‡ | **.406**‡ | **.842**‡ | **.654**‡ | **.803**‡ | **.662**‡ | .519‡ | **.636**‡ |

Table 6: Prompt phrasing ablation for zero-shot PromptReps multi-rep retrieval: MRR@10 for MS MARCO dev and NDCG@10 for TREC DL. D/S/H denote dense, sparse, and hybrid scores. **Bold**: better phrasing for each backbone and column, including rounded ties. Paired two-sided t-tests compare the two phrasings; significant pairs carry † ($p < 0.05$) or ‡ ($p < 0.01$).

| | | MS MARCO dev | | | DL19 | | | DL20 | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Backbone | Phrasing | D | S | H | D | S | H | D | S | H |
| LLaMA3-8B | "three words" | .094‡ | .194 | .181‡ | .216‡ | .416 | .388‡ | .203‡ | **.416** | .407‡ |
| | "a few words" | **.151**‡ | **.196** | **.220**‡ | **.319**‡ | **.454** | **.473**‡ | **.336**‡ | .412 | **.471**‡ |
| Qwen2.5-7B | "three words" | .142 | **.200**‡ | **.211**‡ | .383 | **.445**† | **.485** | **.390** | **.436**† | **.495** |
| | "a few words" | .142 | .177‡ | .200‡ | **.404** | .405† | .477 | .386 | .410† | .478 |

global batch size of 128 (see §4.5).

## B.2 Latency Scaling

The main-paper Figure 1 reports per-query latency at one operating point: the zero-shot autoregressive multi-representation cap $N$=20 on a 100K-document MS MARCO sample. This subsection extends the measurement across two axes: encoding latency as a function of input sequence length, and search latency as a function of index size. Measurements use a single H100 (96GB) GPU with the same attention implementation across all backbones. Runs use synthetic random-token inputs and random-vector indices to isolate systems cost from retrieval correctness.

Figure 5 reports both axes for the four main-paper backbones, in two retrieval interfaces (PromptReps and DiffRetriever), at single- and multi-representation configurations. The multi-representation configurations use the fine-tuned cap $N$=4 for autoregressive backbones (§4.5) and the train-selected $(K_q^*, K_p^*)$ for diffusion backbones (Dream $(4, 16)$, LLaDA $(4, 4)$).

**Encoding scales mildly with input length on both families.** On the encoding axis (top row of Figure 5), all systems scale roughly linearly with input length, but at different absolute levels. Autoregressive multi-representation retrieval sits at 60 to 80 ms across the range; autoregressive single-representation retrieval sits at 15 to 30 ms. DiffRetriever with multiple representations and DiffRe-

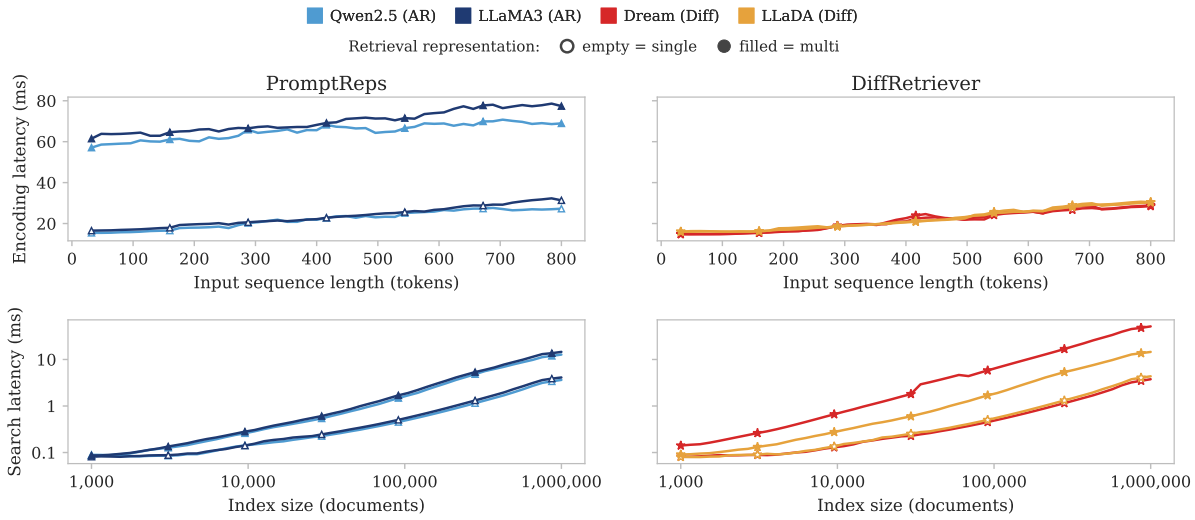


Figure 5: Latency scaling on synthetic inputs and indices. Top row: encoding latency vs. input sequence length (tokens). Bottom row: search latency vs. index size (documents, log scale). Left column: PromptReps on autoregressive backbones (Qwen2.5, LLaMA3). Right column: DiffRetriever on diffusion backbones (Dream, LLaDA). Open markers: single-representation. Filled markers: multi-representation (autoregressive models use $N$=4, the fine-tuned cap; diffusion models use the train-selected $(K_q^*, K_p^*)$). All measurements use a single H100 GPU with the same attention implementation across backbones, 5 warmup queries, and 20 timed runs averaged.

triever with one representation both sit near 15 to 30 ms, with multi-representation overhead of roughly 5 to 10 ms. At $N$=4, the autoregressive-vs.-diffusion encoding ratio is approximately 3× at short inputs and narrows toward longer inputs. The larger ratio in the main paper (≈ 15×) comes from the zero-shot $N$=20 cap, where sequential decoding cost compounds.

**Search cost grows with index size and passage budget.** On the search axis (bottom row of Figure 5), all systems scale predictably with index size under the same search implementation. Multi-representation retrievers have higher search cost than single-representation retrievers because each query performs $K_q$ lookups and the index stores $K_p$ vectors per document. The difference between DiffRetriever with Dream ($K_p$=16) and DiffRetriever with LLaDA ($K_p$=4) is driven by the passage budget: Dream stores 4× as many vectors per document, which translates into higher per-query search cost. At 1M documents, the spread between the slowest multi-representation configuration (≈ 17 ms, DiffRetriever with Dream) and the fastest single-representation configuration (≈ 4 ms) remains smaller than the encoding-side gap.

**Implication.** At the fine-tuned autoregressive cap of $N$=4, the encoding advantage of DiffRetriever over PromptReps narrows relative to the zero-shot $N$=20 setting, but DiffRetriever multi-representation encoding remains close to single-representation encoding, while autoregressive multi-representation encoding remains 2–3× slower than autoregressive single-representation encoding. Within the measured range up to 1M documents, the encoding-side gap remains the dominant contributor to end-to-end latency. For larger deployments, the passage index of multi-representation retrieval, especially Dream's $K_p$=16, may become a more substantial component of total cost.

### B.3 Index Storage Scaling

Multi-representation retrieval changes the storage footprint of the passage index because each passage stores $K_p$ retrieval representations instead of one. We therefore measure index storage as a function of corpus size.

Figure 6 shows that storage grows approximately linearly with the number of indexed documents and with the passage budget $K_p$. Single-representation systems have the smallest footprint, while multi-representation systems require proportionally more storage. This is the main systems tradeoff: DiffRetriever reduces the sequential encoding cost of producing multiple query representations, but it does not remove the cost of indexing multiple passage representations.

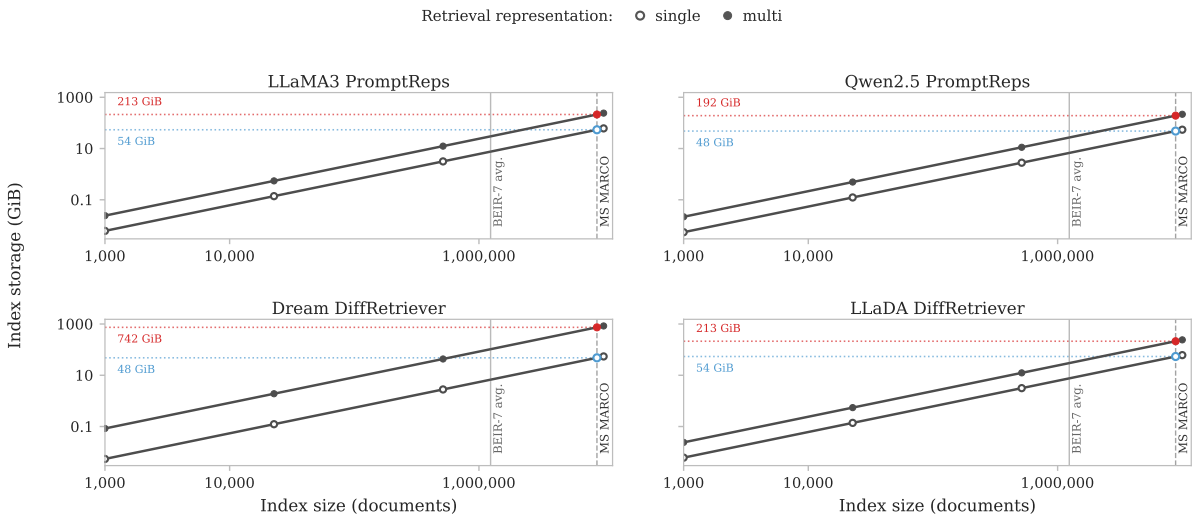


Figure 6: Index storage scaling as a function of corpus size. Open markers denote single-representation variants and filled markers denote multi-representation variants. Vertical reference lines mark the average BEIR-7 corpus size and the MS MARCO passage corpus size. Horizontal dashed lines mark the corresponding MS MARCO-scale storage for single- and multi-representation variants. Storage grows approximately linearly with corpus size and with the number of passage representations $K_p$.

At MS MARCO scale, LLaDA-based DiffRetriever with $(K_q, K_p) = (4, 4)$ requires about 213 GiB for the multi-representation index, compared with about 54 GiB for the corresponding single-representation index. Dream-based DiffRetriever uses a larger passage budget, $(K_q, K_p) = (4, 16)$, and requires about 742 GiB, compared with about 47 GiB for its single-representation variant.

These results clarify the scope of the efficiency claim. The main advantage of DiffRetriever is that multiple retrieval representations are produced in one model forward pass, avoiding the sequential decoding cost of autoregressive multi-representation retrievers. At the same time, the passage-index storage cost can be substantial when $K_p$ is large. This cost remains below token-level late-interaction systems such as ColBERT because DiffRetriever stores a fixed number of representations per passage rather than one per token.

**Reducing the storage cost with compression.** Standard index-compression techniques from the late-interaction literature apply directly to DiffRetriever because each passage stores a fixed number of dense vectors. We follow the ColBERTv2 / PLAID recipe (Santhanam et al., 2022b,a). First, every indexed vector is decomposed into a centroid, assigned by $k$-means clustering over all indexed vectors, plus a residual that captures its offset from that centroid; only the centroid id and the residual are stored, not the original vector. Second, the residual is quantized to a small number of bits per dimension (2 bits in our setup, matching the ColBERTv2 default), which is where most of the storage saving comes from. Third, at search time, an approximate nearest-neighbor lookup over the centroids restricts MaxSim scoring to passages whose vectors lie in the most query-relevant clusters; for those candidates, vectors are reconstructed from centroid + dequantized residual and scored against the query as in uncompressed MaxSim. The result is a compact index that supports the same scoring function used throughout the paper.

Table 8 reports a preliminary check on MS MARCO, TREC DL, and BEIR-7, applying the same compression scheme to all single- and multi-representation DiffRetriever indexes. Across backbones, datasets, and budgets, storage shrinks by roughly $6\times$ (e.g., Dream multi-representation on MS MARCO dev: 742 GiB $\rightarrow$ 118 GiB; LLaDA multi-representation on MS MARCO dev: 213 GiB $\rightarrow$ 34 GiB), with modest effectiveness changes: most paired comparisons differ by no more than $\sim$0.01–0.03 in the relevant metric, and several BEIR-7 datasets (DL19, DL20, SciFact) show no significant drop. A small number of cases, notably LLaDA on Quora, even improve slightly after compression. These results indicate that the headline storage numbers above are an upper bound on deployed cost: the multi-representation index can be compressed by roughly $6\times$ with limited

Table 7: Fine-tuning hyperparameters. Applied uniformly to all four backbones and re-trained baselines.

| | |
|---|---|
| **Training** | |
| Optimizer | AdamW |
| Learning rate | $1\times10^{-4}$ |
| Warmup ratio | 0.06 |
| Epochs | 1 |
| Precision | bf16 |
| Distributed / hardware | ZeRO-2, 4×H100 |
| Random seed | 42 |
| GPU hours | 10-15 hours per model |
| **Batching** | |
| Per-device batch | 8 |
| Grad. accumulation | 4 |
| Global batch | 128 |
| Negatives per query | 1 positive + 15 hard |
| Query / passage length | 32 / 156 tokens |
| InfoNCE temperature | $\tau = 0.01$ |
| **LoRA** | |
| Rank $r$ | 16 |
| Scaling $\alpha$ | 64 |
| Dropout | 0.05 |
| Trainable params | 40–42 M |
| Fraction | 0.52–0.53% |
| **Method-specific** | |
| Diffusion $S$ at training | 1 |
| Dream $(K_q, K_p)$ | $(4, 16)$ |
| LLaDA $(K_q, K_p)$ | $(4, 4)$ |
| PromptReps generation (train) | $N$=4 |
| PromptReps generation (zero-shot) | $N$=20 |
| Decoding temperature | 0.0 |
| **Model Backbone Checkpoint** | |
| Dream-org/Dream-v0-Instruct-7B (Dream) | |
| GSAI-ML/LLaDA-8B-Instruct (LLaDA) | |
| meta-llama/Meta-Llama-3-8B-Instruct (LLaMA3) | |
| Qwen/Qwen2.5-7B-Instruct (Qwen2.5) | |

effectiveness loss, narrowing the practical gap to single-representation retrieval. We leave a more thorough study of compression-aware training and indexing to future work.

# C Supplementary Results

This section reports additional result breakdowns that support the main comparisons in §5.

## C.1 Additional Baseline Comparison

Table 9 reports a broader comparison against dense retrieval baselines beyond the matched-recipe baselines used in the main paper. We group the baselines into BERT-era dense retrievers (Contriever, TAS-B, ColBERT-v2) and recent LLM-based dedicated embedding models (the Qwen3-Embedding line, NV-Embed-v2, GritLM-7B, and PPLX-Embed-4B).

Table 8: Compressed DiffRetriever storage/effectiveness. Eff. and Storage show original→compressed; effectiveness is MRR@10 for MS MARCO Dev and NDCG@10 otherwise. $^*$ marks a paired t-test difference between original and compressed per-query scores ($p < 0.05$).

| Dataset | Backbone | Variant | Eff. | Storage | Ratio |
|---|---|---|---|---|---|
| Dev | Dream | single-rep | .424→.412$^*$ | 48G→7.5G | 6.4× |
| | Dream | multi-rep | .433→.422$^*$ | 742G→118G | 6.3× |
| | LLaDA | single-rep | .424→.406$^*$ | 54G→8.5G | 6.4× |
| | LLaDA | multi-rep | .427→.415$^*$ | 213G→34G | 6.3× |
| DL19 | Dream | single-rep | .741→.729 | 48G→7.5G | 6.4× |
| | Dream | multi-rep | .751→.734$^*$ | 742G→118G | 6.3× |
| | LLaDA | single-rep | .715→.712 | 54G→8.5G | 6.4× |
| | LLaDA | multi-rep | .718→.716 | 213G→34G | 6.3× |
| DL20 | Dream | single-rep | .721→.706 | 48G→7.5G | 6.4× |
| | Dream | multi-rep | .729→.722 | 742G→118G | 6.3× |
| | LLaDA | single-rep | .715→.712 | 54G→8.5G | 6.4× |
| | LLaDA | multi-rep | .721→.732 | 213G→34G | 6.3× |
| NQ | Dream | single-rep | .619→.591$^*$ | 15G→2.3G | 6.5× |
| | Dream | multi-rep | .644→.642 | 225G→36G | 6.3× |
| | LLaDA | single-rep | .620→.606$^*$ | 17G→2.6G | 6.4× |
| | LLaDA | multi-rep | .622→.609$^*$ | 65G→11G | 6.3× |
| HQA | Dream | single-rep | .650→.622$^*$ | 29G→4.4G | 6.4× |
| | Dream | multi-rep | .683→.674$^*$ | 439G→70G | 6.3× |
| | LLaDA | single-rep | .640→.618$^*$ | 32G→5.1G | 6.4× |
| | LLaDA | multi-rep | .647→.605$^*$ | 126G→20G | 6.3× |
| SciFact | Dream | single-rep | .739→.729 | 30M→6.3M | 4.7× |
| | Dream | multi-rep | .752→.741 | 446M→73M | 6.1× |
| | LLaDA | single-rep | .733→.720 | 34M→7.2M | 4.7× |
| | LLaDA | multi-rep | .744→.740 | 129M→23M | 5.8× |
| COVID | Dream | single-rep | .841→.815$^*$ | 962M→150M | 6.5× |
| | Dream | multi-rep | .847→.835 | 15G→2.3G | 6.3× |
| | LLaDA | single-rep | .840→.811$^*$ | 1.1G→171M | 6.4× |
| | LLaDA | multi-rep | .846→.822$^*$ | 4.2G→663M | 6.4× |
| FiQA | Dream | single-rep | .463→.447$^*$ | 320M→52M | 6.2× |
| | Dream | multi-rep | .479→.469$^*$ | 4.9G→790M | 6.3× |
| | LLaDA | single-rep | .453→.443$^*$ | 364M→59M | 6.2× |
| | LLaDA | multi-rep | .443→.437 | 1.4G→228M | 6.2× |
| ArguAna | Dream | single-rep | .406→.405 | 49M→9.3M | 5.3× |
| | Dream | multi-rep | .403→.393$^*$ | 743M→120M | 6.2× |
| | LLaDA | single-rep | .414→.400$^*$ | 56M→11M | 5.3× |
| | LLaDA | multi-rep | .412→.407$^*$ | 214M→36M | 6.0× |
| Quora | Dream | single-rep | .859→.858 | 2.8G→451M | 6.3× |
| | Dream | multi-rep | .887→.885$^*$ | 44G→6.7G | 6.6× |
| | LLaDA | single-rep | .799→.821$^*$ | 3.2G→515M | 6.2× |
| | LLaDA | multi-rep | .798→.831$^*$ | 13G→2.0G | 6.3× |

The LLM-based embedding models are not controlled comparisons for the DiffRetriever retrieval mechanism. They are dedicated embedding models trained on substantially broader retrieval data, including multi-dataset mixtures, MTEB-style instruction-tuning data, synthetic queries, curated hard negatives, and multi-stage training pipelines. DiffRetriever, in contrast, is fine-tuned only on MS MARCO Tevatron triples (§4.5). We therefore report these models as landscape context rather than matched baselines.

Three patterns emerge. First, in the in-domain MS MARCO dev setting, multi-representation DiffRetriever with Dream achieves the highest

Table 9: Dense-only comparison with reported dense baselines and fine-tuned DiffRetriever variants. Variant denotes retrieval representation count. In the MS MARCO group, Dev uses MRR@10 and DL19/DL20 use NDCG@10; BEIR-7 uses NDCG@10. Contriever, TAS-B, and ColBERT-v2 use published scores.

| Method | Backbone | Variant | MS MARCO | | | BEIR-7 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Dev | DL19 | DL20 | NQ | HQA | SciFact | COVID | FiQA | ArguAna | Quora | Avg |
| Contriever | BERT | single-rep | .341 | .678 | .661 | .498 | .638 | .677 | .596 | .329 | .446 | .865 | .578 |
| TAS-B | BERT | single-rep | .340 | .712 | .693 | .465 | .584 | .644 | .505 | .296 | .436 | .835 | .538 |
| ColBERT-v2 | BERT | multi-rep | .397 | .744 | .740 | .562 | .667 | .693 | .738 | .356 | .463 | .852 | .619 |
| Qwen3-Embed-0.6B | Qwen3-0.6B | single-rep | .315 | .662 | .669 | .535 | .651 | .702 | .883 | .458 | .481 | .869 | .654 |
| Qwen3-Embed-4B | Qwen3-4B | single-rep | .357 | .750 | .732 | .632 | .717 | .768 | .884 | .585 | .504 | .876 | .710 |
| Qwen3-Embed-8B | Qwen3-8B | single-rep | .369 | **.757** | .740 | .652 | .737 | .788 | **.929** | .611 | **.524** | .877 | .731 |
| NV-Embed-v2 | Mistral-7B | single-rep | .390 | .751 | **.764** | **.729** | **.823** | **.807** | .886 | **.649** | .478 | .886 | **.751** |
| GritLM-7B | Mistral-7B | single-rep | .343 | .703 | .708 | .599 | .766 | .781 | .743 | .588 | .435 | **.892** | .686 |
| PPLX-Embed-4B | PPLX | single-rep | .364 | .717 | .735 | .632 | .737 | .746 | .857 | .574 | .487 | .887 | .703 |
| **DiffRetriever** | Dream | single-rep | .424 | .741 | .721 | .619 | .650 | .739 | .841 | .463 | .406 | .859 | .654 |
| | | multi-rep | **.433** | .751 | .729 | .644 | .683 | .752 | .847 | .479 | .403 | .887 | .671 |
| | LLaDA | single-rep | .424 | .715 | .715 | .620 | .640 | .733 | .840 | .453 | .414 | .799 | .643 |
| | | multi-rep | .427 | .718 | .721 | .622 | .647 | .744 | .846 | .443 | .412 | .798 | .645 |

score (.433), exceeding every dedicated embedding baseline in the table, including NV-Embed-v2 (.390), Qwen3-Embedding-8B (.369), and Qwen3-Embedding-4B (.357). Second, DiffRetriever substantially outperforms all BERT-era dense retrievers on the BEIR-7 average, including ColBERT-v2 (.619 vs. Dream multi .671). Third, on BEIR-7, dedicated embedding models trained on broader retrieval mixtures, such as Qwen3-Embedding and NV-Embed-v2, achieve higher averages than DiffRetriever, consistent with their broader out-of-domain training coverage. Fine-tuning DiffRetriever on broader retrieval mixtures is a natural next step.

### C.2 Full BEIR-7 Breakdown: Dense, Sparse, and Hybrid

The main-paper Table 3 reports zero-shot results in hybrid mode and fine-tuned results in dense mode, following the scoring-mode shift observed in §5.2. Without supervision, dense alone is less reliable than hybrid scoring; after contrastive fine-tuning, dense scores improve enough that dense alone usually exceeds hybrid. Table 10 reports the full BEIR-7 breakdown across dense, sparse, and hybrid scoring for every system in both settings.

The in-domain mode flip transfers cleanly to BEIR-7. In the zero-shot half, hybrid exceeds dense alone on every system on the BEIR-7 average, by $0.020$ to $0.120$ NDCG@10. In the fine-tuned half, dense exceeds hybrid on every system on the BEIR-7 average, by $0.019$ to $0.028$ NDCG@10. This supports the main-paper choice to report hybrid for zero-shot transfer and dense for fine-tuned transfer. The per-dataset breakdown also shows where individual datasets favor sparse-heavy behavior, such as COVID and Quora for some configurations.

## D Supplementary Analysis

This section provides additional analyses of representation-budget selection and adaptive-budget headroom.

### D.1 MS MARCO Train $(K_q, K_p)$ Grid

Figure 7 shows the zero-shot hybrid retrieval grid on MS MARCO train used for budget selection (§4.4). The star in each panel marks the argmax cell, defining the train-selected $(K_q^*, K_p^*)$ values used unchanged across evaluations: $(4, 16)$ for Dream and $(4, 4)$ for LLaDA.

### D.2 Per-Dataset $(K_q, K_p)$ Landscape

The main-paper Figure 3 reports the $(K_q, K_p)$ landscape on MS MARCO dev and the BEIR-7 average. Here, we expand this view to individual datasets: in-domain DL19 and DL20 (Figure 8), and the seven BEIR-7 datasets (Figure 9). Both figures also include the corresponding aggregate panel from the main paper as a within-figure reference.

The in-domain panels show a consistent pattern. For Dream, the train-selected $(4, 16)$ lands at the per-panel best on every in-domain dataset. For LLaDA, the train-selected $(4, 4)$ lands at the per-panel best on DL19, and on the other two in-domain panels it lies within a flat plateau where multiple cells tie. The per-backbone asymmetry from §6.2—Dream passage-heavy, LLaDA more isotropic around the symmetric center—holds in every in-domain panel.

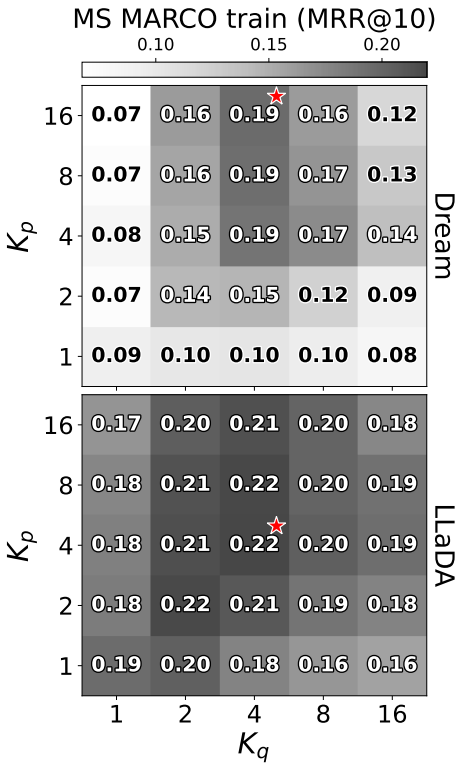


Figure 7: Zero-shot hybrid retrieval grid on MS MARCO train, used for budget selection. Stars mark the argmax cell that defines the fixed $(K_q^*, K_p^*)$ applied unchanged across all evaluations: $(4, 16)$ for Dream and $(4, 4)$ for LLaDA.

The out-of-domain picture is more varied. The train-selected cell lands at the best cell on the BEIR-7 average panel for both backbones, but per-dataset peaks scatter across the grid. HotpotQA peaks at $(8, 16)$ for both backbones, consistent with multi-hop QA benefiting from larger query and passage budgets. LLaDA on TREC-COVID peaks at $(1, 1)$, while LLaDA on ArguAna and Quora peaks at the $K_p = 1$ row, suggesting that additional passage representations help less for these datasets. The train-selected cell is rarely more than .02 below the per-panel best on most datasets, but the preferred budgets differ substantially.

This variance is consistent with the per-query oracle analysis in Appendix D.3: a fixed $(K_q^*, K_p^*)$ works well in aggregate but leaves effectiveness on the table when queries or datasets prefer different budgets.

### D.3 Per-Query Oracle Headroom and Predictability

The main experiments use one fixed representation budget $(K_q^*, K_p^*)$ per diffusion backbone. This is the deployable setting used throughout the paper, and Appendix D.2 shows that these fixed budgets transfer well in aggregate. This subsection asks how much effectiveness could be gained if the budget were chosen separately for each query, and whether query-side budget preferences can be predicted from cheap pre-encoding features.

**Oracle setup.** Figure 10 reports three zero-shot oracles over the same budget grid used in the main analysis. The $K_q$-only oracle adapts $K_q$ per query while holding $K_p$ fixed at $K_p^*$; the $K_p$-only oracle does the converse; and the joint oracle adapts both $K_q$ and $K_p$. These oracles use retrieval labels to choose the best budget for each query, so they are upper bounds rather than deployable systems.

**Oracle headroom.** The joint oracle gives large gains over the fixed zero-shot budget and even exceeds fixed-budget contrastive fine-tuning. On MS MARCO dev, it exceeds the fine-tuned fixed-budget model by .05 MRR@10 for both Dream and LLaDA. On BEIR-7, it exceeds fine-tuning by .03 NDCG@10 for Dream and .07 for LLaDA. Since the oracle uses the frozen zero-shot backbone, the improvement comes from choosing a better budget for each query, not from changing model parameters.

**Deployability.** Not all oracle headroom is equally actionable. Query-side adaptation is the most practical case because the query is encoded online, so a predictor can choose $K_q$ before running the model. Passage-side adaptation is harder because passages are indexed offline: a per-query choice of $K_p$ would require separate passage indexes for each candidate $K_p$, while a more deployable alternative would choose $K_p$ per passage at indexing time. Our current fine-tuning fixes $(K_q, K_p)$ for all examples, so a practical adaptive system would also need training under variable budgets.

**Are query-side preferences predictable?** We next ask whether the query-side oracle preference is structured enough to predict before encoding. Figure 11 correlates the oracle-optimal query budget $K_q^\star$ with two cheap query features: query length and query Shannon entropy. For each query in the pooled MS MARCO dev, DL19, DL20, and BEIR-7 evaluation sets, $K_q^\star$ is the query budget that maximizes the per-query retrieval metric under the corresponding scoring mode. Query length is the model-tokenizer subword count, and query entropy is the Shannon entropy of the query token

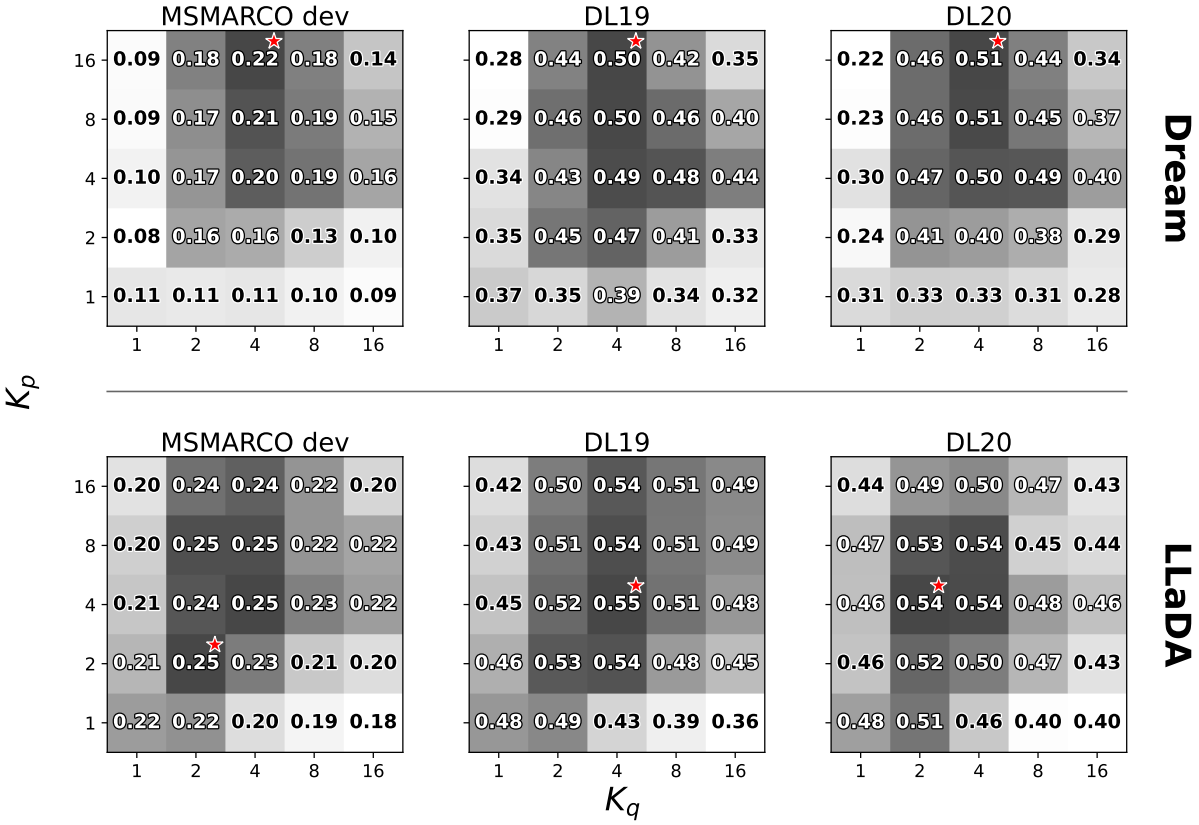


Figure 8: In-domain per-dataset zero-shot hybrid retrieval landscape on MS MARCO dev (MRR@10), TREC DL19, and TREC DL20 (NDCG@10), across $(K_q, K_p) \in \{1, 2, 4, 8, 16\}^2$, for Dream (top row) and LLaDA (bottom row). Stars mark the per-panel best-performing cell. The train-selected $(K_q^*, K_p^*)$ used in the main paper is $(4, 16)$ for Dream and $(4, 4)$ for LLaDA. MS MARCO dev reproduces the corresponding panel from the main-paper Figure 3.

distribution. We report Spearman $\rho$ and Kendall $\tau$ across the pooled set.[6]

**Feature correlations.** Both features correlate positively with the oracle-preferred query budget. Query length is the stronger signal: under dense scoring, it reaches $\rho = +0.31$ for Dream and $\rho = +0.29$ for LLaDA, while under sparse scoring it drops to $\rho = +0.23$ and $\rho = +0.13$. Query entropy shows the same trend but with slightly weaker correlations: $\rho = +0.28$ and $\rho = +0.25$ under dense scoring, and $\rho = +0.19$ and $\rho = +0.11$ under sparse scoring. On the dense panels, peak $K_q^\star$ rises monotonically from approximately 2 for the shortest queries to approximately 8 for the longest. The sparse relationship is flatter, especially for LLaDA, suggesting that sparse signals saturate at smaller query budgets.

**Implication.** These correlations are modest, so length and entropy alone would not recover the full oracle headroom. They nevertheless show that oracle budget preferences are partially structured rather than pure noise. Since both features are available before encoding, they provide a simple starting point for learned query-side budget selection. A practical adaptive retriever would need to learn this selector, decide how to handle passage-side budgets at indexing time, and train the retriever under variable budgets. We leave these steps to future work.

[6]The query-length analysis pools 9 datasets, excluding one dataset whose query field uses a different preprocessing format; the entropy analysis pools the full BEIR-7 plus in-domain evaluation sets (10 datasets total).

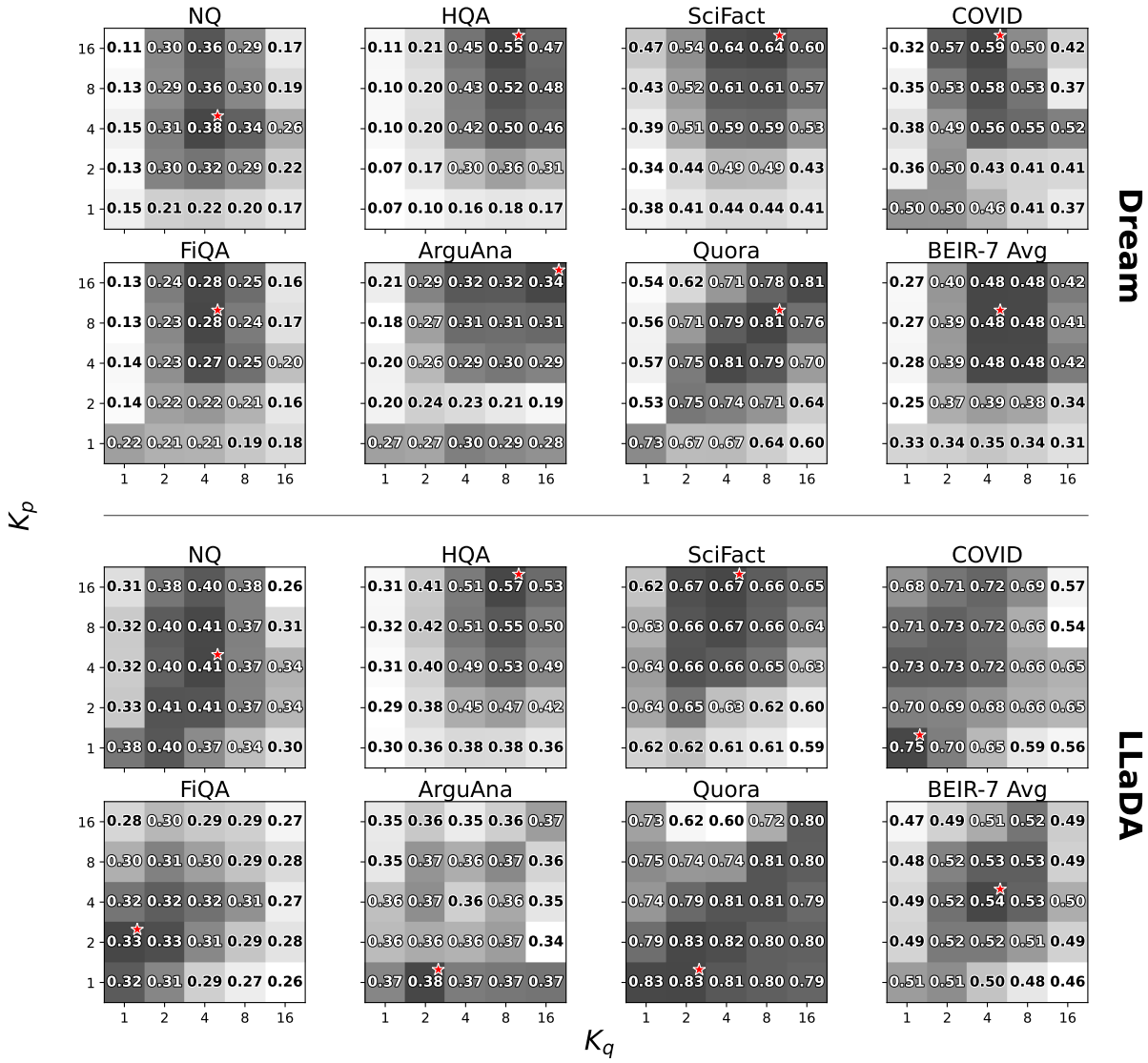


Figure 9: Out-of-domain per-dataset zero-shot hybrid retrieval landscape on the seven BEIR-7 datasets and their average (NDCG@10), across $(K_q, K_p) \in \{1, 2, 4, 8, 16\}^2$, for Dream (top two rows) and LLaDA (bottom two rows). Stars mark the per-panel best-performing cell. The train-selected $(K_q^*, K_p^*)$ used in the main paper is $(4, 16)$ for Dream and $(4, 4)$ for LLaDA. The Avg panel reproduces the BEIR-7 average from the main-paper Figure 3 as a within-figure reference.

Table 10: **Appendix.** BEIR-7 D/S/H results: NDCG@10 dense, sparse, and hybrid scores for every system in Table 3. Fwd denotes query-side forward steps. **Bold**: best within each half and column, including rounded ties. †: better than same-backbone single-rep; ‡: DiffRetriever multi-rep better than corresponding PromptReps single-rep (Dream→Qwen2.5, LLaDA→LLaMA3). Paired t-test, $p < 0.05$.

| | Method | Backbone | Variant | Fwd | NQ | | | HQA | | | SciFact | | | COVID | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | D | S | H | D | S | H | D | S | H | D | S | H |
| Zero-shot | BM25 | — | sparse | — | — | .243 | — | — | **.567** | — | — | **.664** | — | — | .530 | — |
| | DiffEmbed | Dream | single-rep | 1 | .118 | — | — | .149 | — | — | .383 | — | — | .316 | — | — |
| | | LLaDA | single-rep | 1 | .066 | — | — | .167 | — | — | .385 | — | — | .407 | — | — |
| | PromptReps | Qwen2.5 | single-rep | 1 | .156 | .261 | .302 | .066 | .394 | .322 | .455 | .570 | .618 | .490 | .545 | .577 |
| | | | multi-rep | ≤20 | .232† | .265 | .316† | .267† | .484† | .470† | .489 | .595† | .609 | .411 | .513 | .538 |
| | | LLaMA3 | single-rep | 1 | .331 | .293 | .410 | .199 | .452 | .454 | .517 | .584 | .643 | .612 | .558 | .645 |
| | | | multi-rep | ≤20 | .301 | .276 | .379 | .287† | .508† | **.507**† | .539 | .605 | .639 | .530 | .552 | .613 |
| | **DiffRetriever** | Dream | single-rep | 1 | .117 | .102 | .153 | .038 | .056 | .066 | .334 | .276 | .380 | .501 | .339 | .497 |
| | | | multi-rep | 1 | .346†‡ | .219† | .358†‡ | .294†‡ | .411†‡ | .449†‡ | .554†‡ | .564† | .636† | .622†‡ | .481† | .590† |
| | | LLaDA | single-rep | 1 | .289 | **.296** | .382 | .129 | .322 | .299 | .525 | .576 | .618 | **.679** | .666 | **.752** |
| | | | multi-rep | 1 | **.385**†‡ | .292 | **.415**† | **.328**†‡ | .468†‡ | .494†‡ | **.592**†‡ | .624†‡ | **.660**† | .662 | **.690**‡ | .717‡ |
| Fine-tuned | RepLLaMA | LLaMA3 | single-rep | 1 | .622 | — | — | .503 | — | — | .653 | — | — | .564 | — | — |
| | DiffEmbed | Dream | single-rep | 1 | .625 | — | — | .668 | — | — | .735 | — | — | .758 | — | — |
| | | LLaDA | single-rep | 1 | .587 | — | — | .601 | — | — | .714 | — | — | .578 | — | — |
| | PromptReps | Qwen2.5 | single-rep | 1 | .619 | .441 | .573 | .661 | .580 | .684 | .756 | .662 | .727 | .843 | .632 | .797 |
| | | | multi-rep | ≤4 | .623 | .453† | .582† | .686† | .601† | .701† | **.773**† | .681† | **.748**† | **.848** | .664 | .818 |
| | | LLaMA3 | single-rep | 1 | .619 | .443 | .573 | .685 | .595 | .695 | .750 | .659 | .731 | .815 | .596 | .756 |
| | | | multi-rep | ≤4 | .631† | **.458**† | .583† | **.696**† | **.613**† | **.709**† | .735 | .673 | .737 | .809 | .659† | .792† |
| | **DiffRetriever** | Dream | single-rep | 1 | .619 | .433 | .572 | .650 | .587 | .684 | .739 | .670 | .729 | .841 | .629 | .789 |
| | | | multi-rep | 1 | **.644**†‡ | **.458**†‡ | **.596**†‡ | .683†‡ | .603†‡ | .705†‡ | .752 | .666 | .729 | .847 | .665 | **.830**† |
| | | LLaDA | single-rep | 1 | .620 | .446 | .579 | .640 | .595 | .674 | .733 | .681 | .743 | .840 | .691 | .823 |
| | | | multi-rep | 1 | .622 | .452†‡ | .584‡ | .647† | **.613**†‡ | .687† | .744 | **.695**†‡ | .746 | .846 | **.710**‡ | .819‡ |

| | Method | Backbone | Variant | Fwd | FiQA | | | ArguAna | | | Quora | | | Avg | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | D | S | H | D | S | H | D | S | H | D | S | H |
| Zero-shot | BM25 | — | sparse | — | — | **.236** | — | — | **.275** | — | — | **.789** | — | — | **.472** | — |
| | DiffEmbed | Dream | single-rep | 1 | .143 | — | — | .308 | — | — | .685 | — | — | .300 | — | — |
| | | LLaDA | single-rep | 1 | .117 | — | — | .326 | — | — | .528 | — | — | .285 | — | — |
| | PromptReps | Qwen2.5 | single-rep | 1 | .200 | .193 | .271 | .189 | .179 | .209 | .693 | .667 | .784 | .321 | .401 | .441 |
| | | | multi-rep | ≤20 | .165 | .188 | .239 | .231† | .121 | .240† | .600 | .634 | .707 | .342† | .400† | .446† |
| | | LLaMA3 | single-rep | 1 | .272 | .205 | .318 | .229 | .173 | .248 | .729 | .682 | .804 | .412 | .421 | .503 |
| | | | multi-rep | ≤20 | .212 | .207 | .270 | .315† | .133 | .316† | .538 | .686 | .772 | .389 | .424† | .500 |
| | **DiffRetriever** | Dream | single-rep | 1 | .200 | .118 | .220 | .290 | .139 | .273 | .702 | .503 | .725 | .311 | .219 | .331 |
| | | | multi-rep | 1 | .293†‡ | .173† | .278† | .350†‡ | .137 | .323†‡ | .532 | .647† | .696 | .427†‡ | .376† | .476†‡ |
| | | LLaDA | single-rep | 1 | **.311** | .218 | **.325** | .379 | .237 | **.370** | .811 | .706 | **.825** | .446 | .432 | .510 |
| | | | multi-rep | 1 | .308‡ | .225‡ | .317 | **.386**‡ | .179 | .357‡ | **.819**†‡ | .678 | .814‡ | **.497**†‡ | .451†‡ | **.539**†‡ |
| Fine-tuned | RepLLaMA | LLaMA3 | single-rep | 1 | .383 | — | — | .411 | — | — | .821 | — | — | .565 | — | — |
| | DiffEmbed | Dream | single-rep | 1 | .478 | — | — | .375 | — | — | .831 | — | — | .638 | — | — |
| | | LLaDA | single-rep | 1 | .455 | — | — | .387 | — | — | .842 | — | — | .595 | — | — |
| | PromptReps | Qwen2.5 | single-rep | 1 | .444 | .299 | .411 | .407 | .300 | .386 | .872 | .727 | .844 | .657 | .520 | .632 |
| | | | multi-rep | ≤4 | .440 | .293 | .404 | .407 | .287 | .381 | .872 | .729 | .844 | .664† | .530† | .640† |
| | | LLaMA3 | single-rep | 1 | .417 | .268 | .378 | .414 | **.311** | .391 | .860 | .715 | .834 | .651 | .512 | .623 |
| | | | multi-rep | ≤4 | .437† | .290† | .395† | **.416** | .290 | .390 | .865† | .698 | .827 | .655† | .526 | .633† |
| | **DiffRetriever** | Dream | single-rep | 1 | .463 | .292 | .409 | .406 | .288 | .387 | .859 | .703 | .833 | .654 | .515 | .629 |
| | | | multi-rep | 1 | **.479**†‡ | **.316**†‡ | **.431**†‡ | .403 | .305† | .382 | **.887**†‡ | **.748**†‡ | **.859**†‡ | **.671**†‡ | **.537**†‡ | **.647**†‡ |
| | | LLaDA | single-rep | 1 | .453 | .302 | .406 | .414 | .294 | **.392** | .799 | .552 | .746 | .643 | .509 | .623 |
| | | | multi-rep | 1 | .443‡ | .303‡ | .397‡ | .412 | .291 | .389 | .798 | .595† | .760† | .645 | .523† | .626† |

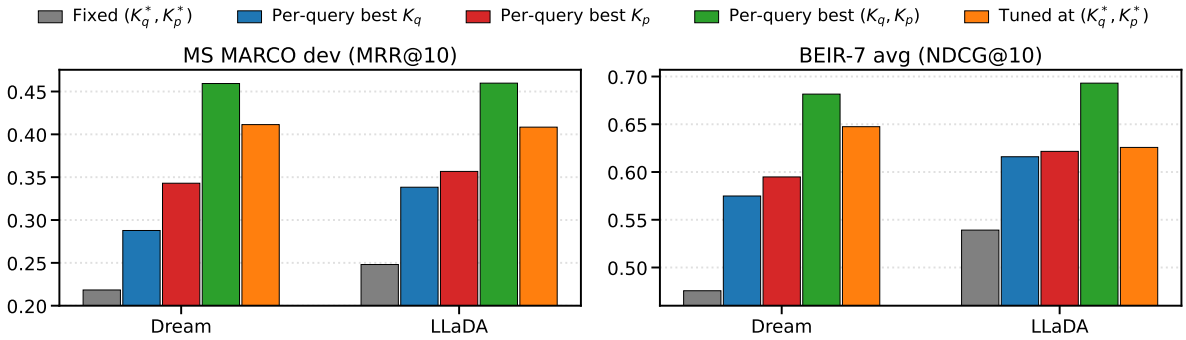


Figure 10: Per-query oracle headroom on MS MARCO dev (MRR@10) and BEIR-7 average (NDCG@10), for Dream and LLaDA. Each group compares the fixed deployable budget $(K_q^*, K_p^*)$ (grey) against three zero-shot per-query oracles adapting $K_q$ only (blue), $K_p$ only (red), or both jointly (green), and the contrastively fine-tuned model at the same fixed budget (orange). Oracles are upper bounds, not deployable systems: they serve as ceilings for learned budget selection on zero-shot DiffRetriever.

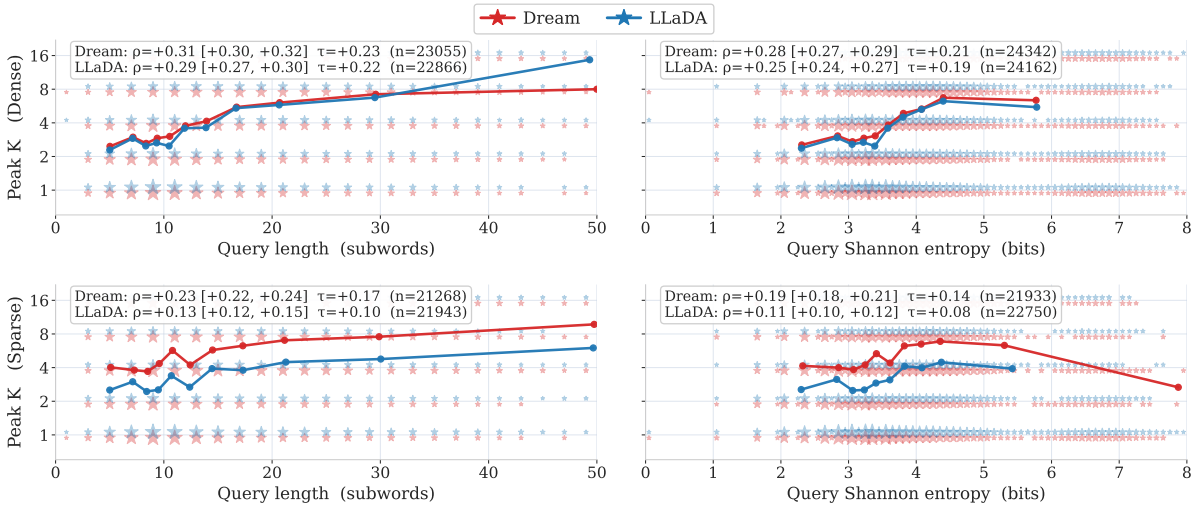


Figure 11: Per-query peak $K_q^\star$ vs. two cheap query features, on Dream and LLaDA. Top row: dense scoring. Bottom row: sparse scoring. Left column: query length (model-tokenizer subwords). Right column: query Shannon entropy (bits, over tokenizer ids). Spearman $\rho$ and Kendall $\tau$ are shown in each panel inset, with 95% bootstrap confidence intervals. Both features correlate positively with peak $K_q$ on both backbones; length is the stronger signal, and dense scoring shows cleaner correlations than sparse.