Compressed Semantic Retrieval as External Memory for Scalable AI Systems

Jiawei Guan [1*] **Yancong Wang** [1*] **Feng Zhang** [1] **Qiang Yin** [2] **Wei Wang** [2] **Xiaoyong Du** [1]

Abstract

Agentic AI systems increasingly treat retrieval as external memory, making storage footprint, index-refresh cost, and query latency first-class scaling constraints. However, standard dense retrieval pipelines decompress corpora into plain text before chunking, embedding, and indexing, inflating storage and build cost whenever memory must be refreshed under tight budgets. In this paper, we present *compressed-domain retrieval*, in which the storage, indexing, and retrieval units coincide at the compressed chunk. The design pairs **ZPKG**, a novel Zstandard-based random-access packaging format, with **CEM**, a byte-level Compressed Embedding Model trained under a joint objective to map compressed chunks into the semantic retrieval space while keeping queries in plain text. Decompression is deferred until after candidate selection, preserving compatibility with standard RAG pipelines. On a six-task MTEB suite, the method reduces corpus storage to 30–33%, cuts indexed vectors by 7.5–10.4×, and accelerates HNSW index construction by 15.1–66.9×. CEM remains competitive with strong text encoders and surpasses all compared baselines on WinoGrande and TempReasonL1, positioning compressed-domain retrieval as a practical external-memory substrate for scaling agentic systems under memory, refresh, and latency budgets.

1. Introduction

Agentic AI systems, including retrieval-augmented assistants (Hui et al., 2024; Borgeaud et al., 2022; Lewis et al., 2020), tool-using agents (Wu et al., 2024b; Yao et al., 2022; Schick et al., 2023), and long-horizon decision pipelines (Goyal et al., 2022; Wang et al., 2023), increasingly rely on retrieval as external memory (Singh et al., 2025). In this setting, memory must be stored, refreshed, and queried under finite resource budgets, so retrieval quality alone is not sufficient: storage footprint, index-refresh cost, and query latency are also first-class constraints, and often dominate end-to-end system cost as memory scales or changes frequently (Cheng et al., 2024; De Jong et al., 2023; Yang et al., 2025).

In large-scale deployments, corpora are typically stored, transferred, and refreshed in compact packaged forms such as compressed shards or archived document bundles (Collet & Kucherawy, 2018; Bartík et al., 2015; Devi Kotha et al., 2019), because these formats reduce storage and movement cost. Standard dense retrieval pipelines, however, do not index such compressed objects directly (Izacard et al., 2021). They first restore the corpus to plain text, split it into chunks, embed each chunk, and then build a nearest-neighbor index (Jayaram Subramanya et al., 2019). As a result, whenever external memory is maintained in compressed form, semantic indexing still proceeds through a decompression-first workflow over an expanded text representation.

This observation motivates the central question of this work: *must semantic indexing always be built on decompressed text?* We argue it need not. This paper explores a different design point, in which compressed text itself becomes the substrate for semantic retrieval. The key conceptual shift is to treat compression not as a storage-layer optimization beneath an otherwise unchanged retriever, but as part of the retrieval representation itself. Under this view, compressed chunks are no longer intermediate artifacts that must be expanded before semantic access; they become the primary memory objects that the system stores, indexes, retrieves, and decompresses only when needed downstream.

Enabling semantic retrieval directly over compressed text raises three coupled challenges. First, *representation mismatch*: standard retrievers are defined over plain-text chunks, whereas compressed storage exposes byte sequences whose local structure diverges from ordinary linguistic patterns. Second, *granularity alignment*: for compressed text to serve as a retrieval-native substrate, the unit at which data is stored must coincide with the unit at which it is indexed and re-

[*]Equal contribution [1]Renmin University of China [2]China Electronics Technology Kingbase (Beijing) Technologies Inc.. Correspondence to: Feng Zhang <fengzhang@ruc.edu.cn>.

Proceedings of the 43rd International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).

trieved, rather than being re-segmented after expansion. Third, *selective recovery*: the system must preserve efficient random access so that only retrieved segments are decompressed, rather than requiring the corpus to be materialized in advance. Together, these challenges show that compression cannot be left beneath the retriever as an implementation detail; it must be designed into the retrieval representation.

We therefore present a ***compressed-domain retrieval architecture*** in which storage, indexing, and retrieval all operate over the same compressed representation. To address representation mismatch, we develop the **Compressed Embedding Model (CEM)**, a byte-level encoder trained with teacher–student alignment to map compressed chunks into the semantic space induced by a standard text encoder, while queries remain plain text. To enforce granularity alignment, we introduce **ZPKG**, a Zstandard-based random-access packaging format that makes the compressed chunk the shared unit of packaging, indexing, and retrieval. To support selective recovery, ZPKG provides offset-based localization so that decompression is deferred until after semantic candidate selection. The resulting pipeline changes the indexing substrate from decompressed text to compressed memory objects while remaining compatible with conventional downstream generation modules.

This design opens a new operating point for agentic retrieval systems: external memory can be stored and indexed directly in compressed form, without materializing an expanded text view. On a six-task MTEB suite, it reduces corpus storage to 30–33% of raw size, cuts indexed vectors by 7.5–10.4×, and accelerates HNSW index construction by 15.1–66.9×—gains that matter most when memory is refreshed, transferred, or rebuilt under tight budgets. At the same time, the compressed representation remains semantically usable rather than merely archival: CEM is competitive with strong text encoders overall and surpasses all compared baselines on WinoGrande and TempReasonL1.

We summarize our contributions as follows: (i) **Compressed-domain retrieval**, a new external-memory design that treats compressed text as a retrieval-native representation rather than a storage-side artifact; (ii) **A compression-aligned retrieval architecture** that unifies storage, indexing, and retrieval over compressed memory objects, combining CEM for semantic encoding with ZPKG for random-access packaging and late decompression; and (iii) **Empirical evidence** on the MTEB benchmark demonstrating a strong quality–efficiency trade-off while preserving useful semantic retrieval capability.

2. Related Work

Retrieval as external memory. A growing line of work treats retrieval as a form of non-parametric external memory for language models and agentic systems (Kang et al., 2025; Tan et al., 2025). Retrieval-augmented generation (RAG) uses retrieved evidence to ground generation (Lewis et al., 2020), RETRO (Borgeaud et al., 2022) integrates large-scale retrieval into language-model training and inference, and LongMem (Wang et al., 2023) studies persistent memory access through a retriever-reader architecture. More recent evaluations further argue that such memory systems should be judged not only by retrieval accuracy, but also by the cost of building, maintaining, and accessing memory in end-to-end use (Wu et al., 2024a). Our work is aligned with this systems view of retrieval memory, but addresses a different question: *how to reduce the storage, indexing, and refresh overhead of the memory substrate itself*.

Dense retrieval and semantic indexing. Modern dense retrieval methods represent queries and documents in a shared embedding space, substantially improving semantic matching over purely lexical retrieval (Karpukhin et al., 2020; Zhang et al., 2025; Robertson & Zaragoza, 2009; Wang et al., 2024). However, dense retrieval systems are typically built over decompressed text units: documents are first materialized as text, segmented into chunks, embedded, and then indexed for approximate nearest-neighbor search. In this pipeline, compression may reduce storage or transfer cost upstream, but the semantic index is still defined over an expanded text representation. Our setting preserves the standard text-query interface of dense retrieval, but changes the indexed object itself: *instead of embedding restored text chunks, we embed directly addressable compressed chunks*.

Compression in retrieval systems. Prior work has explored search or processing over compressed text, showing that useful operations can sometimes be supported without fully materializing the original text (Gilbert et al., 2023; Guan et al., 2023; 2024; Chen et al., 2023). More broadly, compression has long been used as a systems optimization for storage, transport, and execution efficiency. Our goal is different. We do not treat compression as a lower-layer optimization beneath an otherwise unchanged retriever. Instead, we make compression part of the retrieval representation: *the stored object, indexed object, and retrieved object are all the same compressed chunk, with decompression deferred until after candidate selection*. This design shifts compression from an implementation detail to a first-class component of the retrieval pipeline.

Byte-level representation learning. Direct semantic access over compressed chunks requires models that can operate on raw byte sequences rather than tokenized natural language. Byte-level architectures are a natural starting point because they avoid tokenizer mismatch and can model structure di-

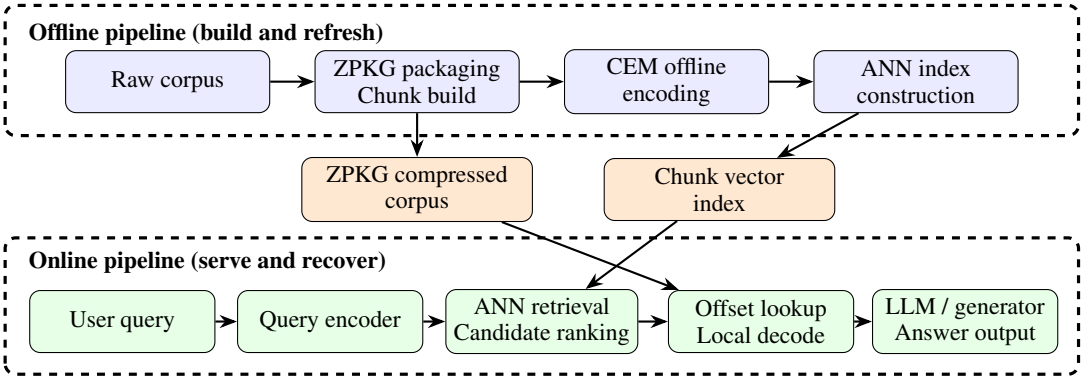

Figure 1. End-to-end architecture of compressed-domain retrieval.

rectly in byte streams. Recent work such as BLT (Pagnoni et al., 2025) shows that byte-level modeling with latent patching can scale effectively. Our Compressed Embedding Model builds on this direction, but targets a different objective: *rather than modeling generic byte sequences, it learns retrieval embeddings for losslessly compressed chunks under text-side semantic supervision, so that compressed documents can be retrieved using standard plain-text queries*.

3. Methodology

3.1. Design Principle and Problem Formulation

Our design rests on a single principle: the storage unit, the indexing unit, and the retrieval unit should coincide. Standard dense retrieval materializes a decompressed view purely to support semantic indexing, even when the corpus is stored and transferred in compressed form. We discard this intermediate view and treat a *compressed chunk*—not a decompressed text span—as the primary retrieval object. Queries remain plain text; documents are packaged, indexed, retrieved, and only then locally decompressed.

This principle is motivated by three deployment constraints that dominate the cost of external memory at scale: (i) *storage footprint*, which limits how much memory can be maintained or transported; (ii) *refresh cost*, which determines how quickly memory can be rebuilt after updates; and (iii) *tail latency*, which compounds across the repeated retrieval calls issued inside an agentic loop.

Formally, given a query q and a corpus $\mathcal{C} = \{d_i\}_{i=1}^{N}$, a packaging function Π maps the corpus into our specially designed ZPKG shards $\mathcal{P} = \{P_r\}_{r=1}^{R}$. Each shard P_r contains a local dictionary Δ_r, a sequence of compressed bytes $\{b_{r,i}\}_{i=0}^{n_r-1}$, and a tail offset table I_r. The basic retrieval unit is a compressed chunk identified by the composite key (r, i), where r denotes the shard and i the chunk index within that shard; the same address is reused for retrieval, localization, and selective decompression. Let $g(\cdot)$ denote the text query encoder, $f(\cdot)$ the compressed-chunk encoder, and $s(\cdot,\cdot)$ cosine similarity in the shared retrieval space. For a query q, retrieval returns the top-k compressed chunk identifiers

$$\mathcal{R}_k(q) = \mathrm{TopK}_{(r,i)} \; s\big(g(q), f(b_{r,i})\big), \tag{1}$$

and only the selected chunks are locally decompressed.

3.2. Architecture Overview

Figure 1 gives the end-to-end architecture. The system has an offline build path and an online retrieval path. Offline, the raw corpus is packaged into ZPKG shards, each compressed chunk is encoded by our Compressed Embedding Model, and an ANN index is built over the resulting chunk vectors. Online, the plain-text query is mapped into the same retrieval space and used to retrieve top-k chunk identifiers. Only the selected chunks are then localized through shard offsets and locally decompressed for downstream use.

The pipeline keeps the compressed chunk as the common systems unit throughout: ZPKG defines and addresses that unit at the storage layer, and CEM maps it into the semantic retrieval space. Storage, indexing, and retrieval therefore all operate over the same object, rather than over a decompressed surrogate created solely to support indexing. This joint design directly addresses the three challenges raised in Section 1—CEM for representation mismatch, ZPKG for granularity alignment, and offset-based localization for selective recovery—while preserving a drop-in interface for conventional RAG and agent pipelines.

3.3. ZPKG Data Format Design

ZPKG realizes granularity alignment and selective recovery by making the compressed chunk the shared unit of storage, indexing, and retrieval. As shown in Figure 2, each package consists of four regions: a fixed-length *header*, a shard-local *dictionary section*, a sequence of compressed *chunk frames*, and a tail *index table*.

The *header* stores package metadata, including format version, chunk count, and the starting offset of the index table.

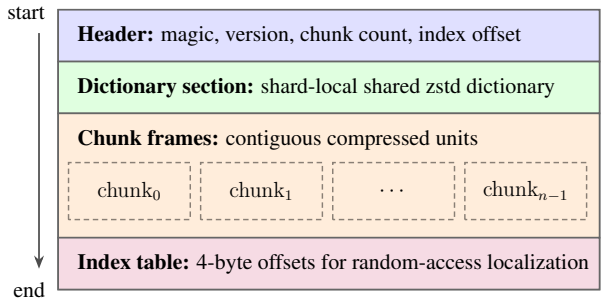

Figure 2. ZPKG package layout. Each package is self-contained and comprises four regions.

The *dictionary section* stores the shared Zstandard dictionary used within the package. We make the dictionary shard-local, rather than corpus-wide, so that each package remains self-contained and can be moved or refreshed independently of a global codec artifact. The *chunk frames* define the retrieval units. ZPKG uses *output-driven chunking* in compressed space rather than token-based chunking in decompressed text, because the quantities that dominate systems cost—storage footprint, transfer volume, and decompression work—are measured in compressed bytes. Compression proceeds as a stream, and a frame is sealed once its compressed size reaches a target threshold $\tau < s_t$, where s_t is the nominal chunk size and the gap $s_t - \tau$ acts as a guard margin that limits overshoot at finalization. The *index table* stores the starting byte offset of every chunk frame, making each frame directly addressable. Let s_i denote the stored start offset of chunk i. If the tail index starts at byte offset b_I and stores 4-byte offsets, then the address of the i-th entry is $a_i = b_I + 4i$, and reading adjacent stored offsets yields the compressed span

$$\ell_i = \begin{cases} s_{i+1} - s_i, & i < n-1, \\ b_I - s_i, & i = n-1. \end{cases} \tag{2}$$

This layout supports constant-time localization of retrieved chunks and keeps decompression as a strictly post-retrieval operation. Because ANN vectors are produced one-for-one from compressed chunks, packaging and indexing remain directly aligned, and no redundant re-chunking step is needed.

3.4. Compressed Embedding Model (CEM)

CEM addresses the representation-mismatch challenge by learning retrieval vectors directly from losslessly compressed chunks. Although compression preserves the underlying content exactly, it exposes that content as byte sequences whose structure—dictionary references, back-references, entropy-coded symbols—is no longer explicit in surface text form. CEM's role is therefore to recover a retrieval-compatible semantic representation from compressed bytes while keeping queries in standard text form.

Architecture. We build CEM on a byte-level architecture inspired by Byte Latent Transformer (Pagnoni et al., 2025). As shown in Figure 3, the retrieval-side stack has three stages before the pooling head: a byte-local encoder, entropy-based patching, and a latent transformer.

The byte-local encoder operates directly on the raw compressed byte stream and produces contextualized byte representations through short-range attention, without relying on a text tokenizer. Byte-local processing is a deliberate choice for the compressed domain: it avoids the tokenizer mismatch that would arise from feeding compressed bytes into a subword vocabulary, and it keeps the model sensitive to the local byte-level regularities (dictionary references, frame boundaries, entropy-coded symbols) that carry much of the structure in compressed streams. Entropy-based patching then groups neighboring byte representations into variable-length *patches* according to the local predictability of the byte stream. Regions of low entropy—regular, repetitive byte patterns—are summarized into coarser patches, while information-dense regions, such as boundaries or rare symbol runs, retain finer resolution. This adaptive granularity matches computational cost to information content, which is particularly valuable on compressed byte sequences whose information density is highly non-uniform. The latent transformer then performs global self-attention over the resulting patch sequence, yielding latent states $\{h_j\}_{j=1}^{P}$ that capture chunk-level semantics. A pooling-and-projection head maps these states into the shared retrieval space:

$$z = \mathrm{Norm}\left(W \cdot \frac{1}{P} \sum_{j=1}^{P} h_j \right), \tag{3}$$

where W is a learned projection and $\mathrm{Norm}(\cdot)$ enforces unit-norm embeddings for cosine-similarity search.

These three stages let CEM model compressed chunks as structured byte sequences rather than as corrupted text, while keeping the downstream retrieval interface identical to that of a standard text encoder.

Training objective. To align compressed-domain representations with the semantic space used by text queries, we train CEM with a joint objective combining cross-domain distillation and retrieval discrimination. A frozen teacher text encoder consumes the decompressed text of each chunk, while the student CEM consumes the compressed bytes directly. The total loss is:

$$\mathcal{L} = \lambda_1 \tilde{\mathcal{L}}_{\text{distill}} + \lambda_2 \tilde{\mathcal{L}}_{\text{cd}} + \lambda_3 \tilde{\mathcal{L}}_{\text{InfoNCE}} + \lambda_4 \tilde{\mathcal{L}}_{\text{triplet}}, \tag{4}$$

where each term plays a distinct role. The *pointwise distillation loss* $\tilde{\mathcal{L}}_{\text{distill}}$ pulls each student embedding toward the teacher embedding of the same chunk on a per-example basis. The *contrastive distillation loss* $\tilde{\mathcal{L}}_{\text{cd}}$ complements this by matching pairwise similarity structure between student

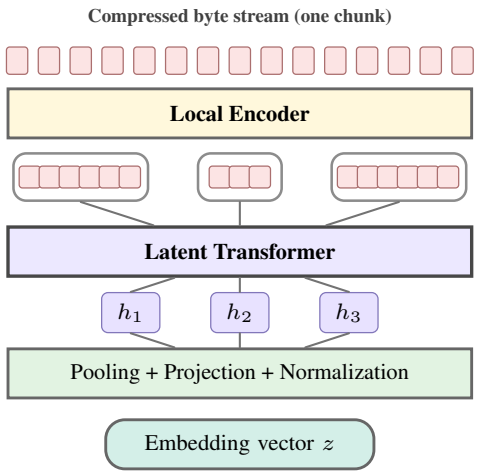

Figure 3. Compressed Embedding Model (CEM). Compressed bytes are locally encoded, grouped into three entropy-adaptive patches, processed by a latent transformer into three latent states, and pooled into a single retrieval embedding.

and teacher embeddings across a batch, preserving relative semantic neighborhoods rather than only absolute positions. The *InfoNCE loss* and *triplet loss* then sharpen query-conditioned separation between relevant and non-relevant chunks, using in-batch and hard negatives respectively.

Training data. Training operates at the chunk level rather than the document level. Starting from supervised query–document relevance pairs, we package each relevant document into ZPKG chunks and treat the resulting chunks as positives for the associated query. Negatives are drawn both from random cross-shard samples and from hard negatives mined as false retrieval hits. This setup produces compressed-domain embeddings that are simultaneously aligned with the text-side semantic space and discriminative under query-conditioned retrieval.

4. Experiments

4.1. Experimental Setup

Evaluation suite. We evaluate the proposed method on six datasets drawn from MTEB (Muennighoff et al., 2023): AILAStatutes, LEMBPasskeyRetrieval, SpartQA, TempReasonL1, TwitterHjerneRetrieval, and WinoGrande. The same suite is used for both systems measurements and retrieval quality measurements so that storage footprint, embedding throughput, index construction cost, and retrieval quality are reported under a single, consistent experimental setting.

Models and baselines. The query encoder and teacher text encoder are both Qwen3-Embedding-0.6B (Zhang et al., 2025). For systems comparisons we evaluate CEM against Qwen3-0.6B and BGE-M3; for the broader retrieval-quality comparison we additionally include E5-large/base/small and GTE-large/base/small.

Training and indexing. CEM outputs 1024-dimensional embeddings and is trained with AdamW (learning rate 5×10^{-5}, 10 epochs, 1000 warmup steps). The four loss terms in Eq. 4 use stage-dependent weights $\lambda_1 \in [0.5, 1.0]$, $\lambda_2 \in [0.2, 1.0]$, $\lambda_3 \in [0.1, 1.0]$, and $\lambda_4 \in [0.1, 0.5]$ across early, middle, and late phases. The InfoNCE temperature is fixed at 0.07. ZPKG uses shard-local dictionaries with a maximum size of 64KB. All experiments reported below use 2KB chunks. During offline encoding, text encoders use batch size 64, whereas CEM uses batch size 8, reflecting the larger information content per compressed chunk. All ANN indexes use cosine similarity with a uniform HNSW configuration: `m=16`, `ef_construction=64`, and `ef_search=80`.

Hardware and software. Experiments run on a server equipped with an NVIDIA A40 46GB GPU, Intel Xeon Gold 5318Y CPUs (2 sockets, 48 cores / 96 threads), and 251GB RAM. The software stack is PyTorch 2.6.0 with CUDA 12.4, PostgreSQL 12.20, and pgvector.

Metrics. We report storage footprint, embedding throughput (docs/s), HNSW index build time (s), and retrieval quality (nDCG@10, reported$\times$100).

4.2. Overall Results

Table 1 summarizes the four evaluation axes together, and three observations stand out. First, **compressed-domain retrieval delivers consistent systems gains across the pipeline**. CEM reduces storage footprint relative to the text-side baselines on every dataset in the suite and achieves the lowest HNSW build time on every dataset in the summary table. Second, **these gains do not come at the cost of semantic usefulness**. Although embedding throughput varies by dataset, CEM is particularly strong on SpartQA, TempReasonL1, and WinoGrande, where it combines the highest encoding throughput with the best retrieval quality. Third, **the quality results place compressed-domain retrieval inside the competitive frontier rather than below it**. Qwen3-Embedding-0.6B leads on AILAStatutes and LEMBPasskeyRetrieval, BGE-M3 leads on TwitterHjerneRetrieval, and CEM leads on SpartQA, TempReasonL1, and WinoGrande. Taken together, these results show that compressed-domain retrieval is not a lower-cost approximation of text-side indexing: it defines a favorable quality–efficiency frontier and supports compressed chunks as a practical, retrieval-native memory representation.

These findings are obtained under a modest experimental budget—short training schedule and single-node hardware. Even in this regime CEM already achieves the best retrieval quality on three of six tasks while delivering substantial

Table 1. Overall evaluation on six MTEB tasks. Storage reports CEM against the shared Qwen/BGE text-side representation; throughput measured in docs/s, index build time in seconds, and retrieval quality is nDCG@10×100. Qwen denotes Qwen3-Embedding-0.6B and BGE denotes BGE-M3.

	Storage		Embedding Throughput (docs/s)			Index Build Time (s)			Retrieval Quality		
Dataset	CEM	Qwen/BGE	CEM	Qwen	BGE	CEM	Qwen	BGE	CEM	Qwen	BGE
AILAStatutes	91 KB	162 KB	24.76	7.19	2.46	0.002	0.037	0.018	50.67	74.19	29.19
LEMBPasskeyRetrieval	305 KB	22.12 MB	88.84	14.12	7.10	0.001	0.027	0.022	43.90	84.75	59.00
SpartQA	15 KB	136 KB	1094.43	574.53	975.36	0.000	0.024	0.022	43.749	8.302	7.492
TempReasonL1	102 KB	527 KB	2890.98	942.72	1704.59	0.002	6.89	8.46	18.71	1.022	0.99
TwitterHjerneRetrieval	32 KB	41 KB	178.92	423.79	362.49	0.002	0.074	0.075	52.71	60.04	71.43
WinoGrande	45 KB	212 KB	2164.82	847.89	1684.87	0.001	2.56	2.94	98.53	50.792	41.72

systems advantages across the rest of the suite, suggesting that the reported numbers are a conservative lower bound on what compressed-domain retrieval can achieve at scale.

4.3. Storage Footprint

We first examine storage cost. In Table 1, Qwen3-Embedding-0.6B and BGE-M3 share a single Qwen/BGE column because both operate over the same corpus representation. Across all six datasets, aligning the storage, indexing, and retrieval units around compressed chunks reduces the footprint of the external-memory substrate, but the magnitude of the reduction is governed by the corpus compression ratio that ZPKG's shard-local dictionary and entropy coder can achieve on that corpus. Two datasets illustrate the range. On LEMBPasskeyRetrieval, 712 KB of source text yields a shard-local dictionary of only 377 B—3% of the final package—and compression delivers a roughly two-orders-of-magnitude reduction in footprint. On TwitterHjerneRetrieval, by contrast, a 41 KB source corpus produces a 21.7 KB dictionary that alone accounts for 68% of the 32 KB package, and this fixed overhead bounds the net benefit on short, low-redundancy corpora. Compressibility is therefore the single corpus-level quantity that determines where ZPKG's storage gain is largest, and a systematic study across length distributions, redundancy profiles, and domain diversity is a natural extension of this analysis.

4.4. Embedding Throughput

Table 2 reports embedding throughput, extending the comparison to $E5_{\text{large}}$ and GTE_{large}. CEM achieves the highest throughput on LEMBPasskeyRetrieval, SpartQA, TempReasonL1, and WinoGrande, while GTE_{large} is fastest on AILAStatutes and TwitterHjerneRetrieval. Throughput and storage are coupled under compressed-domain retrieval. Because CEM performs one forward pass per compressed chunk rather than per tokenized passage, its document-level throughput scales with the same corpus compression ratio identified in Section 4.3. On highly compressible corpora—LEMBPasskeyRetrieval, TempReasonL1, and WinoGrande—each chunk replaces many tokenized passages and CEM's advantage widens despite its smaller batch size (8 vs. 64). TwitterHjerneRetrieval lies at the opposite end, dominated by dictionary overhead; AILAStatutes is intermediate, with documents that are neither short (mean 1973.6 characters, median 899) nor highly repetitive, and therefore admits only modest compression.

Compressibility further predicts where CEM's efficiency and retrieval-quality advantages coincide. It typically reflects strong local structure, a regime naturally suited to byte-level modeling, and on TempReasonL1 and WinoGrande the two axes reinforce each other. The alignment is not universal—LEMBPasskeyRetrieval favors CEM in throughput but Qwen3-Embedding-0.6B in retrieval quality—but it indicates that compressibility is a corpus-level signal predictive of when compressed-domain retrieval pays off on efficiency and quality simultaneously.

Table 2. Embedding throughput (docs/s) on six MTEB tasks.

Dataset	CEM	Qwen3-0.6B	BGE-M3	$E5_{\text{large}}$	GTE_{large}
AILAStatutes	24.76	7.19	2.46	28.32	**112.24**
LEMBPasskeyRetrieval	**88.84**	14.12	7.10	20.34	88.32
SpartQA	**1094.43**	574.53	975.36	995.36	562.69
TempReasonL1	**2890.98**	942.72	1704.59	776.94	1236.68
TwitterHjerneRetrieval	178.92	423.79	362.49	135.56	**466.21**
WinoGrande	**2164.82**	847.89	1684.87	845.63	1220.38

4.5. Index Construction

Table 3. HNSW index build time (seconds) on six MTEB tasks.

Dataset	CEM	Qwen3-0.6B	BGE-M3	$E5_{\text{large}}$	GTE_{large}
AILAStatutes	**0.002**	0.037	0.018	0.025	0.016
LEMBPasskeyRetrieval	**0.001**	0.027	0.022	0.010	0.008
SpartQA	**0.000**	0.024	0.022	0.023	0.021
TempReasonL1	**0.002**	6.89	8.46	5.938	8.253
TwitterHjerneRetrieval	**0.002**	0.074	0.075	0.076	0.082
WinoGrande	**0.001**	2.56	2.94	2.961	3.102

Table 3 reports HNSW index build time and also extends the comparison to $E5_{\text{large}}$ and GTE_{large}. Index construction shows the clearest single-metric advantage for the proposed method: **CEM achieves the lowest build time on**

Table 4. Retrieval quality (nDCG@10×100) on six MTEB tasks against nine public embedding baselines. **Bold** marks the best score per task, underline the second-best; "#Best" counts the number of tasks on which each model attains the top score.

Task	CEM	Qwen3-0.6B	BGE-M3	$E5_{large}$	$E5_{base}$	$E5_{small}$	GTE_{large}	GTE_{base}	GTE_{small}
AILAStatutes	50.67	**74.19**	29.19	20.17	17.26	20.44	23.32	22.30	25.02
LEMBPasskeyRetrieval	43.90	**84.75**	59.00	38.50	36.25	37.25	38.00	31.00	35.00
SpartQA	**43.749**	8.302	7.492	3.084	4.637	3.316	1.696	2.879	2.969
TempReasonL1	**18.71**	1.022	0.99	1.68	1.37	1.74	1.62	1.22	1.54
TwitterHjerneRetrieval	52.71	60.04	**71.43**	43.43	39.83	35.21	43.18	37.27	35.55
WinoGrande	**98.53**	50.792	41.72	26.96	27.26	16.53	17.40	8.35	13.43
Average	**51.38**	46.52	34.97	22.30	21.10	19.08	20.87	17.17	18.92
#Best	**3**	2	1	0	0	0	0	0	0

every dataset in the suite, often by one to three orders of magnitude. This result follows directly from the design of the compressed pipeline. Because ANN vectors are produced one-for-one from compressed chunks—and compressed chunks are substantially larger in source content than token-based chunks—CEM indexes far fewer objects prior to ANN construction, which sharply reduces both graph-building work and the in-memory footprint of the index itself. This advantage is especially important in deployments where external memory is refreshed or rebuilt frequently, since lower build time translates directly into lower maintenance overhead for an agentic system whose memory must track changing data.

4.6. Retrieval Quality

We finally examine whether the systems gains come at the cost of semantic retrieval effectiveness. Table 4 compares CEM against a broader set of public embedding baselines. The results show that compressed-domain retrieval preserves meaningful semantic discrimination and can surpass strong text encoders on selected workloads. CEM attains the highest average nDCG@10 and the best task score on SpartQA, TempReasonL1, and WinoGrande, ahead of every E5 and GTE variant despite operating directly on compressed bytes. Compressed representations therefore need not be viewed only as an efficiency-oriented approximation; they can act as an effective retrieval substrate in their own right. Taken together with the storage, throughput, and index-construction results, these findings support a stronger conclusion: **CEM is not simply a lower-cost replacement for text-side indexing, but a new point in the retrieval design space that offers substantial systems gains without uniformly sacrificing retrieval quality**. This is precisely the trade-off that matters for memory-constrained or frequently refreshed retrieval systems—and, by extension, for agentic pipelines whose end-to-end utility depends jointly on semantic quality and memory-maintenance cost.

5. Discussion and Limitations

Our results identify compressed-domain retrieval as a concrete new operating point for scalable external memory: consistent systems gains across storage, indexing, and refresh cost, paired with retrieval quality that is competitive with state of the art. The limitations below bound the scope of this claim, not its principle. First, CEM is trained under a deliberately modest budget—short schedule, single-node hardware—while the strongest text baselines benefit from substantially larger pretraining corpora and compute. That CEM nevertheless leads on TempReasonL1 and WinoGrande and remains competitive elsewhere suggests the reported numbers are a conservative lower bound on what compressed-domain encoding can achieve at scale. Second, both storage and throughput gains are governed by a single quantity—corpus compressibility—so the efficiency advantage is largest on corpora with rich local structure and smallest on short or low-redundancy corpora, where the shard-local dictionary bears a disproportionate fixed cost. A systematic characterization across length distributions, redundancy profiles, and domains is a natural extension. Third, our evaluation isolates the retrieval substrate; extending the study to end-to-end agentic loops—iterative retrieval, tool use, long-horizon planning—is the logical downstream direction, and one the present results are designed to enable.

6. Conclusion

We introduced *compressed-domain retrieval*, a new external-memory design in which compressed chunks serve directly as the units of storage, indexing, and retrieval. By combining ZPKG, a random-access compressed package format, with CEM, a byte-level encoder aligned to the semantic space of standard text queries, the method integrates compression into the retrieval representation itself while preserving compatibility with conventional RAG pipelines. Experiments on MTEB show clear efficiency gains with competitive retrieval quality, supporting compressed memory as a practical design point for AI systems under storage, refresh, and latency constraints. More broadly, our results

suggest the promise of retrieval and memory pipelines that operate directly on compressed objects rather than on decompressed surrogates.

Acknowledgements

This work is supported by the National Natural Science Foundation of China (No.625B2176), the Qiushi Academic Project of Renmin University of China (NO.RUC24QSDL093), and the Outstanding Innovative Talents Cultivation Funded Programs 2023 of Renmin University of China. Feng Zhang is the corresponding author.

References

Bartík, M., Ubik, S., and Kubalik, P. Lz4 compression algorithm on fpga. In *2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, pp. 179–182. IEEE, 2015.

Borgeaud, S., Mensch, A., Hoffmann, J., Cai, T., Rutherford, E., Millican, K., van den Driessche, G., Lespiau, J.-B., Damoc, B., Clark, A., de Las Casas, D., Guy, A., Menick, J., Ring, R., Hennigan, T., Huang, S., Maggiore, L., Jones, C., Cassirer, A., Brock, A., Paganini, M., Irving, G., Vinyals, O., Osindero, S., Simonyan, K., Rae, J. W., Elsen, E., and Sifre, L. Improving language models by retrieving from trillions of tokens, 2022. URL https://arxiv.org/abs/2112.04426.

Chen, Z., Zhang, F., Guan, J., Zhai, J., Shen, X., Zhang, H., Shu, W., and Du, X. Compressgraph: Efficient parallel graph analytics with rule-based compression. *Proceedings of the ACM on Management of Data*, 1(1):1–31, 2023.

Cheng, X., Wang, X., Zhang, X., Ge, T., Chen, S.-Q., Wei, F., Zhang, H., and Zhao, D. xrag: Extreme context compression for retrieval-augmented generation with one token. *Advances in Neural Information Processing Systems*, 37:109487–109516, 2024.

Collet, Y. and Kucherawy, M. Zstandard compression and the application/zstd media type. Technical report, Internet Engineering Task Force, 2018.

De Jong, M., Zemlyanskiy, Y., FitzGerald, N., Ainslie, J., Sanghai, S., Sha, F., and Cohen, W. W. Pre-computed memory or on-the-fly encoding? a hybrid approach to retrieval augmentation makes the most of your compute. In *International Conference on Machine Learning*, pp. 7329–7342. PMLR, 2023.

Devi Kotha, H., Tummanapally, M., and Upadhyay, V. K. Review on lossless compression techniques. *Journal of Physics: Conference Series*, 1228(1):012007, 2019.

Gilbert, H., Sandborn, M., Schmidt, D. C., Spencer-Smith, J., and White, J. Semantic compression with large language models, 2023. URL https://arxiv.org/abs/2304.12512.

Goyal, A., Friesen, A., Banino, A., Weber, T., Ke, N. R., Badia, A. P., Guez, A., Mirza, M., Humphreys, P. C., Konyushova, K., et al. Retrieval-augmented reinforcement learning. In *International Conference on Machine Learning*, pp. 7740–7765. PMLR, 2022.

Guan, J., Zhang, F., Ma, S., Chen, K., Hu, Y., Chen, Y., Pan, A., and Du, X. Homomorphic compression: Making text processing on compression unlimited. *Proceedings of the ACM on Management of Data*, 1(4):1–28, 2023.

Guan, J., Zhang, F., Tang, Y., Ye, W., and Du, X. Hocopg: A database system with homomorphic compression for text processing. *Proceedings of the VLDB Endowment*, 17(12):4477–4480, 2024.

Hui, Y., Lu, Y., and Zhang, H. Uda: A benchmark suite for retrieval augmented generation in real-world document analysis. *Advances in Neural Information Processing Systems*, 37:67200–67217, 2024.

Izacard, G., Caron, M., Hosseini, L., Riedel, S., Bojanowski, P., Joulin, A., and Grave, E. Unsupervised dense information retrieval with contrastive learning. *arXiv preprint arXiv:2112.09118*, 2021.

Jayaram Subramanya, S., Devvrit, F., Simhadri, H. V., Krishnawamy, R., and Kadekodi, R. Diskann: Fast accurate billion-point nearest neighbor search on a single node. *Advances in neural information processing Systems*, 32, 2019.

Kang, J., Ji, M., Zhao, Z., and Bai, T. Memory OS of AI agent. In Christodoulopoulos, C., Chakraborty, T., Rose, C., and Peng, V. (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 25961–25970, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1318. URL https://aclanthology.org/2025.emnlp-main.1318/.

Karpukhin, V., Oguz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., and Yih, W.-t. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*, pp. 6769–6781, 2020.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.

Muennighoff, N., Tazi, N., Magne, L., and Reimers, N. Mteb: Massive text embedding benchmark. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 2014–2037, 2023.

Pagnoni, A., Pasunuru, R., Rodriguez, P., Nguyen, J., Muller, B., Li, M., Zhou, C., Yu, L., Weston, J. E., Zettlemoyer, L., et al. Byte latent transformer: Patches scale better than tokens. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9238–9258, 2025.

Robertson, S. and Zaragoza, H. *The probabilistic relevance framework: BM25 and beyond*, volume 4. Now Publishers Inc, 2009.

Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., and Scialom, T. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

Singh, A., Ehtesham, A., Kumar, S., and Khoei, T. T. Agentic retrieval-augmented generation: A survey on agentic rag. *arXiv preprint arXiv:2501.09136*, 2025.

Tan, Z., Yan, J., Hsu, I.-H., Han, R., Wang, Z., Le, L., Song, Y., Chen, Y., Palangi, H., Lee, G., Iyer, A. R., Chen, T., Liu, H., Lee, C.-Y., and Pfister, T. In prospect and retrospect: Reflective memory management for long-term personalized dialogue agents. In Che, W., Nabende, J., Shutova, E., and Pilehvar, M. T. (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8416–8439, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.413. URL https://aclanthology.org/2025.acl-long.413/.

Wang, L., Yang, N., Huang, X., Yang, L., Majumder, R., and Wei, F. Improving text embeddings with large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11897–11916, 2024.

Wang, W., Dong, L., Cheng, H., Liu, X., Yan, X., Gao, J., and Wei, F. Augmenting language models with long-term memory. *Advances in Neural Information Processing Systems*, 36:74530–74543, 2023.

Wu, D., Wang, H., Yu, W., Zhang, Y., Chang, K.-W., and Yu, D. Longmemeval: Benchmarking chat assistants on long-term interactive memory. *arXiv preprint arXiv:2410.10813*, 2024a.

Wu, S., Zhao, S., Huang, Q., Huang, K., Yasunaga, M., Cao, K., Ioannidis, V. N., Subbian, K., Leskovec, J., and Zou, J. Avatar: Optimizing llm agents for tool usage via contrastive reasoning. *Advances in Neural Information Processing Systems*, 37:25981–26010, 2024b.

Yang, T., Yao, Z., Jin, B., Cui, L., Li, Y., Wang, G., and Liu, X. Demystifying and enhancing the efficiency of large language model based search agents. *arXiv preprint arXiv:2505.12065*, 2025.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K. R., and Cao, Y. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.

Zhang, Y., Li, M., Long, D., Zhang, X., Lin, H., Yang, B., Xie, P., Yang, A., Liu, D., Lin, J., et al. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025.