Learned Sparse Retrieval with Vector Quantization

Sai Sreenivas Chintha saisreenivas@umass.edu
Durga Prasad Maram dmaram@umass.edu
Harshitha Kolukuluru hkolukuluru@umass.edu
Sairohith Yanamala syanamala@umass.edu

1 Problem Statement

Efficient and effective retrieval of relevant documents is a fundamental challenge in information retrieval (IR). The traditional retrieval methods can be broadly categorized into **lexical matching** and **semantic matching**. Lexical retrieval methods such as BM25, rely mainly on exact keyword matching and leverage inverted indices for efficient search. While these methods are computationally efficient, they often suffer from the **vocabulary mismatch problem**, where relevant documents might be overlooked if they do not contain the exact query terms. On the other hand, **semantic retrieval methods** take advantage of dense embeddings to capture deeper contextual meanings, enabling more robust matching. However, these methods require **high-dimensional vector searches**, making them computationally expensive for accurate large-scale retrieval despite advances like Approximate Nearest Neighbor (ANN) search techniques.

Recent approaches attempt to bridge this gap by learning sparse representations from dense embeddings, where words in the vocabulary are reweighted, dropped, or substituted with related terms to improve retrieval performance. While these methods retain the efficiency of lexical retrieval, they remain constrained by vocabulary-based indexing.In this project, we propose a retrieval framework that learns sparse, descrete representations in a latent semantic space through vector quantization (VQ). Instead of projecting dense embeddings onto a fixed vocabulary or constructing linear combinations of lexical tokens, we discretize chunked dense embeddings by mapping each subvector to its nearest codeword in a learned codebook. This results in compact, interpretable sequences of latent semantic tokens. These symbolic representations can be efficiently indexed and retrieved using classical inverted indexing techniques such as BM25. To further enhance retrieval quality and enable end-to-end learning, we extend the architecture with a sparse transformation layer that projects the quantized embeddings into a high-dimensional sparse space. This allows the model to learn retrieval-aware sparse activations, combining the semantic expressiveness of neural models with the efficiency of symbolic retrieval.

This approach offers several advantages: (1) it captures the semantic relationships at a deeper level while maintaining high retrieval efficiency, (2) it reduces the memory and computational expenses by replacing the dense vector queries with sparse index lookups, and (3) it enables learning topics in semantic space rather than projecting embeddings into the vocabulary (as in current LSR approaches), allowing the model to learn more contextualized, meaningful representations. Through the use of VQ-based sparse representations, this work seeks to leverage the benefits of both neural and lexical IR approaches.

2 What we proposed vs. what we accomplished

Our original proposal primarily aimed to discretize the latent dense embeddings using vector quantization and to use the resulting sequence of codebook cluster indices as the sparse representation. While we implemented this approach (Method A in Section 6), based on proposal feedback, we extended it by projecting the vector-quantized representation of the mean-pooled final layer token embeddings onto a latent vocabulary space (Method B in Section 6) to obtain the sparse representation.

- Collect and preprocess MSMARCO-style passage and query dataset - Completed

- Build and evaluate BM25 and dense retrieval baselines (e.g., Contriever) - Completed
- Implement symbolic retrieval using vector quantization (Method A) and test different codebook configurations- Completed
- Extend to end-to-end sparse projection model over quantized embeddings(Method B) - Completed
- *Make Method B outperform all baselines*: Partially successful: Method B produced the most sparse representation in the latent vocabulary space without significantly degrading performance when vector quantization (VQ) was introduced into the pipeline. When important terms from the latent vocabulary (8 for the query and 32 for the passage) were used to expand the original query and document, the method outperformed BM25 on the development set (search space was a smaller subset of the corpus).

3 Related work

Information retrieval (IR) methods have evolved significantly in recent times, shifting from traditional lexical matching techniques to neural-based semantic matching models.

Traditional IR systems are predominantly based on lexical matching, with methods such as BM25 (Robertson and Zaragoza, 2009) and TF-IDF ranking documents based on strict term match and term frequency-inverse document frequency weight. Such methods are computationally inexpensive owing to their reliance on inverted indexing but they often perform poorly with respect to synonymy and vocabulary mismatch problems (Furnas et al., 1987).

To address these issues, neural IR models were used which use dense representations of text to capture semantic relations between words. Although early approaches like Word2Vec (Mikolov et al., 2013), and GloVe (Pennington et al., 2014) employed word embeddings to do the same, more recent retrieval models rely mostly on contextualized embeddings from transformers such as BERT (Devlin et al., 2019). More notable neural retrieval models include Dense Passage Retrieval (DPR) (Karpukhin et al., 2020), which uses a bi-encoder-based model to represent(embed) queries and documents in the same space, ColBERT (Khattab and Zaharia, 2020), which improves efficiency by computing token-level similarities.

Despite being semantically rich, these dense retrieval models often suffer from high latency and memory overhead issues, requiring the application of Approximate Nearest Neighbor (ANN) search-based techniques like FAISS (Douze et al., 2025) and HNSW (Malkov and Yashunin, 2018) to alleviate the retrieval time. Nevertheless, ANN search remains computationally expensive, especially for large-scale retrieval tasks.

Sparse Representation Learning for Retrieval: In order to combine the efficacy of inverted indexes with the semantic benefits of dense models, more recent works have attempted to generate sparse representations of words from their dense embeddings. These methods mainly try to achieve lexicon-style representations while simultaneously preserving the semantic representations. In this line of work, SPLADE (Formal et al., 2021b) introduced a self-learned expansion procedure where the transformer model predicts sparse weights over the vocabulary tokens, using L1 and FLOPS(Paria et al., 2020) based regularization to enforce sparsity. Deep Impact (Mallia et al., 2021) and uniCOIL (Lin and Ma, 2021) enhance sparse representations by re-weighting term importance rather than simply expanding the queries. In Contriever (Izacard et al., 2022) and retroMAE(Xiao et al., 2022) use contrastive learning and masked autoencoding to enhance sparse retrieval capabilities.

These approaches have proved effective for improving retrieval efficiency but they still rely on mapping the dense embeddings to predefined fixed vocabulary words, which limits their flexibility and generalization capabilities. In this project, we aim to differ from this idea and want to explore the sparsity within the latent space of words rather than in the vocabulary space.

More recently, KALE (Borges et al., 2023) proposed a lightweight method that projects dense embeddings into a k-sparse latent vocabulary. These learned semantic terms improve first-stage retrieval and can complement or replace lexical vocabularies, offering gains in both accuracy and efficiency. Our approach shares this goal of latent-space symbolic representation, but differs by explicitly clustering embedding subspaces and supporting symbolic indexing over quantized codeword sequences.

Vector Quantization for Discrete Representations:
Vector Quantization (VQ) has been an extensively used technique for compression and discrete representation learning and has proven effective in tasks like image generation (VQ-VAE (van den Oord et al., 2018)), speech processing(Baevski et al., 2020), and discrete latent space modeling. The fundamental idea of VQ is to map continuous embeddings into a learned set of discrete codebook vectors, hence reducing the memory requirements while simultaneously preserving valuable information.

Though VQ has found use in generative modeling, its application to retrieval is still mostly untapped. Some studies have investigated clustering-based retrieval such as hierarchical clustering of document embeddings (Xu and Jiang, 2022), but they still do not provide us with structured, tokenizable representations for efficient indexing. Our proposed method leverages a VQ-based codebook learning to create a sequence of cluster indices – a sparse representation in a radically different form.

Clustering based Retrieval:
Clustering-based structures have been explored in the works of topic modeling (e.g., LDA (Blei et al., 2003)) and document clustering (Aggarwal and Zhai, 2012) but these methods however are often not dynamically adaptive and are also not usually intended for large-scale retrieval applications.

Recent works on semantic tokenization like DeepCluster of visual features(Caron et al., 2019) suggest that clustering embeddings into discrete tokens can improve model interpretability. Our work further plans to extend this idea by treating clusters as semantic tokens and using inverted indexing with BM25 over these tokens instead of raw vocabulary words.

4 Dataset

We train and evaluate our models on the **MS MARCO** (Bajaj et al., 2016) passage ranking dataset in the full ranking setting. The corpus consists of approximately 8.8 million passages and hundreds of thousands of training queries, each paired with an average of 1.1 relevant passages. Due to computational constraints, we use a 2 million passage subset for training. To perform contrastive learning, we incorporate pre-mined hard negatives retrieved using BM25, which we obtain from an existing processed version of the dataset available on Hugging Face. For evaluation, we use the official development set, which contains approximately 6,900 queries. More information about the dataset is available at `https://microsoft.github.io/msmarco/Datasets`.

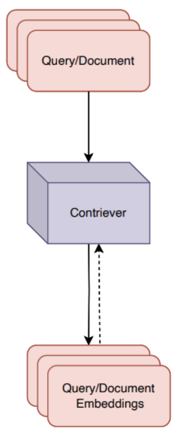
5 Baselines

To validate the effectiveness of our proposed retrieval framework, we compare it against several representative and widely-adopted baselines in the information retrieval literature. These baselines span lexical, dense, and learned sparse retrieval paradigms, allowing us to assess how well our symbolic representation approach bridges the efficiency-effectiveness trade-off. The performance of these baselines on the dev set is shown in Table 1.

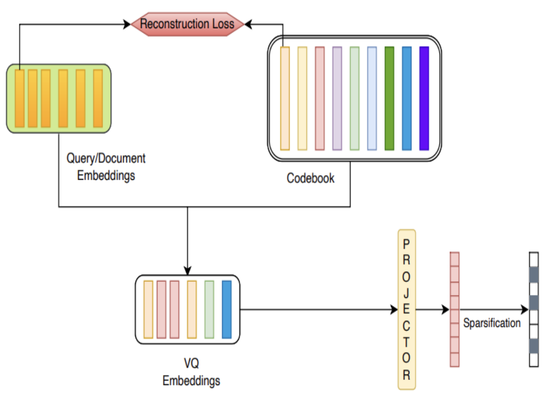
BM25 (Lexical Baseline). We use BM25 as our classical sparse retrieval baseline. BM25 relies on exact term matches and inverse document frequency weighting, and is known for its high efficiency and interpretability. However, it suffers from vocabulary mismatch and lacks semantic generalization. We use default parameters, which we found to perform well on MSMARCO dev in prior work; no tuning was performed on the test set.

Dense Bi-encoder with ANN (Contriever). As a strong dense retrieval baseline, we evaluate a Contriever bi-encoder model (Lei et al., 2023) which was fine-tuned on the MS-MARCO dataset. Queries and passages are encoded independently and compared via dot product. The passage embeddings are indexed using FAISS for approximate nearest neighbor search. This approach captures semantic similarity and often outperforms lexical models, but incurs high storage and runtime costs.

Learned Sparse Retrieval (SPLADE / SPLADEv2). We compare against SPLADE (Formal et al., 2021b) and SPLADEv2 (Formal et al., 2021a), which generate sparse term-weight vectors by expanding vocabulary-based tokens via a masked language model and sparse regularization. These models bridge dense and sparse paradigms and are considered state-of-the-art in learned sparse retrieval.



(a) Dense Encoder



(b) End to End pipeline

Figure 1: LSR_VQ

Baselines				
Model	MRR@10	NDCG@10	R@10	R@1000
BM25	0.86	0.87	0.95	0.99
DPR	0.97	0.97	0.997	0.999
SPLADE*	0.32	-	-	0.95

Table 1: Baselines - Search space consists of ∼ 7.4K passages. *Note that SPLADE is evaluated on ∼ 8.8M passages.

Hyperparameter Tuning and Validation. For all baselines, we avoid any tuning on the test set. Instead, hyperparameters are either taken from public checkpoints or from the respective papers.

Train/Validation/Test Split. We use the existing MS MARCO splits: the training split is used to create separate train and validation sets. The train set is used to train the VQ module, sparse projection, and encoding models, while the validation set is used for hyperparameter tuning. The development set (MS MARCO dev) is used for testing and final evaluation.

6 Our approach

In this section, we present a sparse retrieval framework that operates in a latent semantic space by discretizing dense neural embeddings through Vector Quantization (VQ). Unlike traditional vocabulary-based sparse retrieval models that rely on exact lexical term matches, our method learns a codebook of latent semantic tokens and maps dense embedding chunks to these discrete units. This transformation enables symbolic indexing over semantically meaningful concepts rather than surface-level word representations. The resulting sparse representations preserve the efficiency and interpretability of inverted indexing while capturing the expressiveness of neural encoders.

We begin by detailing the core pipeline, which leverages a bi-encoder architecture to generate dense representations that are quantized into token sequences via a learned codebook. We then describe an extended formulation that jointly optimizes for codebook reconstruction fidelity, representational sparsity, and semantic alignment through a contrastive learning objective, resulting in compact, discriminative, and retrieval-effective representations.

This retrieval framework is instantiated in two complementary forms:

- **Method A: Symbolic Retrieval with Vector Quantization.** This approach discretizes chunked dense embeddings into latent tokens and performs retrieval using symbolic indices such as BM25 over learned codes.
- **Method B: Sparse Projection with Vector Quantization (LSR_VQ).** This model projects quantized embeddings into high-dimensional sparse vectors using a learned transformation, enabling end-to-end training and more expressive scoring via sparse dot products. Refer Figure 1.

In the following sections, we first present the shared dense encoder and quantization mechanism, followed by detailed descriptions of each method. Our source code is available online.[1]

[1]https://github.com/sensai99/LSR-VQ

6.1 Vector Quantization-Based Sparse Retrieval

6.1.1 Dense Representation

We begin by encoding passages and queries independently using a fine-tuned `Contriever` model, a bi-encoder architecture tailored for dense retrieval tasks. Given an input sequence, the model produces contextualized token embeddings from the final layer of the transformer. To obtain a single fixed-length representation, we apply mean pooling across all non-padding token embeddings. Formally, for a sequence of length T, the pooled representation is computed as:

$$\mathbf{h} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{x}_t$$

where $\mathbf{x}_t \in R^d$ is the embedding of the t-th token, and d denotes the final hidden-layer dimensionality of the encoder. This results in a dense vector $\mathbf{h} \in R^d$ for each input (passage or query). We apply this encoding pipeline independently to all passages in the corpus and all queries.

6.1.2 Vector Quantization of Chunked Representations

To introduce finer-grained semantic structure into these dense representations, we partition each embedding vector $\mathbf{h} \in R^d$ into C non-overlapping, contiguous chunks:

$$\mathbf{h} = \left[\mathbf{h}^{(1)}, \mathbf{h}^{(2)}, \ldots, \mathbf{h}^{(C)}\right], \quad \mathbf{h}^{(i)} \in R^{d/C}$$

where each subvector $\mathbf{h}^{(i)}$ represents a localized subspace of the full embedding. This chunking process enables the model to assign multiple latent tokens to a single passage, allowing different semantic regions of the embedding to be discretized independently.

Each of these chunked vectors is subsequently passed through a vector quantization module, which maps it to the nearest codeword in a learnable codebook $\mathcal{E} \in R^{d/C \times K}$ consisting of K latent semantic tokens, each of which is a codeword. Initially, we adopt a purely data-driven approach by initializing the codebook weights from a standard Gaussian distribution. The quantizer is trained with L2 error to move the codebook vectors towards dense embeddings.

Once the dense embeddings are partitioned and the codebook is initialized, we discretize each chunked subvector using a vector quantization module. This module maps continuous representations into a finite set of latent semantic tokens, allowing for symbolic encoding of passages and queries in a learned semantic space. Specifically, given a chunk $\mathbf{h}^{(i)} \in R^{d'}$, the quantizer identifies the nearest codeword from the codebook $\mathcal{E} = \{\mathbf{e}_1, \ldots, \mathbf{e}_K\}$ by minimizing the squared Euclidean distance:

$$\mathcal{Q}(\mathbf{h}^{(i)}) = \mathbf{e}_{j^*}, \quad j^* = \arg\min_j \left\| \mathbf{h}^{(i)} - \mathbf{e}_j \right\|_2^2$$

The quantized embedding for a query/passage is then represented as a sequence of codeword indices (Method A):

$$\mathbf{z} = [j_1, j_2, \ldots, j_C], \quad \mathbf{z} \in \{1, \ldots, K\}^C$$

or its assigned codebook vector (Method B):

$$\mathcal{Q}(\mathbf{h}) = [e_{j1}, e_{j2}, \ldots, e_{jC}]$$

This transformation produces a symbolic representation of each input, with each token reflecting semantic content localized to a subspace of the original dense vector. Unlike traditional vocabulary-based tokenization, these latent tokens are learned directly from data and are not tied to surface-level linguistic units.

The quantizer is implemented using the VQ-VAE-style exponential moving average (EMA) updates, which ensure stable codebook learning. For a codeword $\mathbf{e}_j$ and assignment counts n_j, the updates follow:

$$\begin{aligned} \mathbf{e}_j &\leftarrow \rho \cdot \mathbf{e}_j + (1-\rho) \cdot \bar{\mathbf{h}}_j \\ n_j &\leftarrow \rho \cdot n_j + (1-\rho) \cdot c_j \end{aligned}$$

where $\bar{\mathbf{h}}_j$ is the sum of all chunk vectors assigned to codeword j, c_j is the corresponding count, and ρ is the EMA decay factor.

To prevent codebook collapse, we apply a dead code reinitialization strategy that re-seeds underutilized codewords with randomly selected active chunk embeddings. To improve convergence speed and promote early semantic structure in the codebook, we later incorporate a distillation-inspired initialization strategy. Specifically, we extract all chunked passage embeddings from the corpus and apply MiniBatch **KMeans clustering** to group them into K semantic clusters. The resulting centroids are then used to initialize the

Num-chunks	MRR@10	NDCG@10	R@10	R@1000
8	0.34	0.39	0.57	0.91
12	0.40	0.45	0.63	0.92
16	0.46	**0.50**	**0.66**	**0.92**
24	**0.47**	**0.50**	0.61	0.91

Table 2: Results of Method A using BM25 with 6,000 clusters in vector quantization. Search space consists of $\sim$ 7.4K passages.

Method	MRR@10	NDCG@10	R@10	R@1000
Vector-VQ	0.46	0.50	0.66	0.92
KMeans + Vector-VQ	0.49	0.54	0.7	0.92
Residual VQ	0.49	0.52	0.65	0.95

Table 3: Results of Method A across different vector quantization methods using BM25 with 6,000 clusters. Search space consists of $\sim$ 7.4K passages.

quantizer codebook prior to training. This warm-start provides a meaningful prior over latent token space, reduces cold start variance, and empirically improves both codebook utilization and retrieval performance during early training phases.

6.1.3 Semantic Indexing and Sparse Retrieval

After quantization, each passage and query is represented as a sequence of discrete latent codes drawn from the learned codebook:

$$\mathbf{z}_d = [j_1, \dots, j_C]$$

$$\mathbf{z}_q = [k_1, \dots, k_C], \quad j_i, k_i \in \{1, \dots, K\}$$

These code sequences serve as symbolic surrogates for dense embeddings, capturing semantic content in a compact and interpretable form. They enable sparse retrieval using classical indexing techniques, now applied in a learned latent space.

Once passages and queries are quantized into sequences of latent codeword indices, we construct an inverted index over the codeword indices of passages and apply standard BM25 scoring during retrieval. Specifically, given a query q and passage d, the relevance score is computed as:

$$\text{score}_{\text{BM25}}(q, d) =$$

$$\textstyle\sum_{t \in q \cap d} \text{idf}(t) \cdot \frac{f_{t,d} \cdot (k_1 + 1)}{f_{t,d} + k_1 \cdot \left(1 - b + b \cdot \frac{|d|}{\text{avgdl}}\right)}$$

where $f_{t,d}$ is the frequency of token t in d, and avgdl is the average document length. This formulation enables efficient symbolic retrieval entirely over the learned latent space using classical inverted indexing techniques.

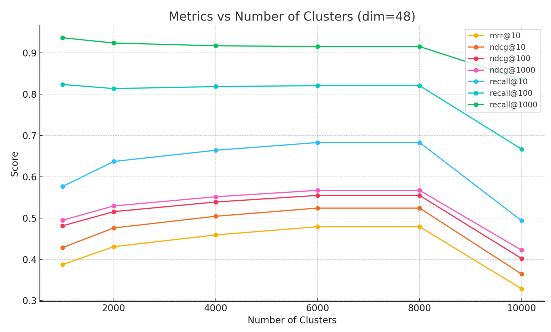


Figure 2: Effect of number of clusters on retrieval metrics.

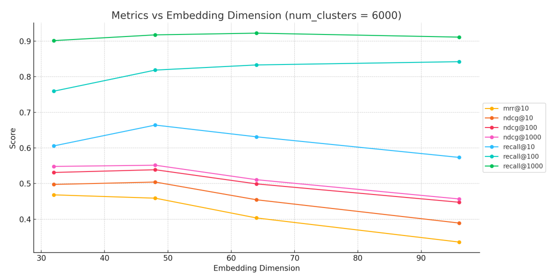


Figure 3: Effect of embedding dimension on retrieval metrics.

6.1.4 Experimentation

We perform controlled ablations across several quantization parameters. Specifically, we vary the codebook dimensionality ($d' \in \{32, 48, 64, 96\}$), the number of codewords ($K \in \{1000, 2000, 4000, 6000, 8000, 10000\}$), and the number of chunks ($C \in \{24, 16, 12, 8\}$). Results (refer Figure 2, Figure 3 & Table 2) indicate retrieval performance improves steadily with more clusters up to 8000, but slightly degrades at 10,000 clusters, suggesting that excessive quantization may reduce semantic coherence. Overall, we achieve best results with codebooks sized $K = 6000$–8000 and moderate chunking ($C = 16$) strike the best balance between sparsity and retrieval fidelity.

We have also observed that initializing with K-Means centroids over chunked embeddings improves performance. We further experimented with another approach known as residual vector quantization (RVQ), where multiple codebooks are used sequentially to refine approximation quality. While RVQ improves reconstruction and provides only small gains in retrieval accuracy, it increases inference overhead due to multi-stage lookup, making it most useful only in low-capacity settings. Refer Table 3.

Due to computational constraints, we restrict the retrieval search space to only the pool of relevant documents for the queries (6,980) in the devset; the results are reported in Table 2 using the best configurations (6000 clusters and 16 chunks for each vector).

6.2 Learned Sparse Retrieval with Vector Quantization

While the above described method enables fast and interpretable symbolic retrieval via latent token indexing, it is naive and less powerful. Now we extend this pipeline into a fully differentiable and end-to-end learnable framework. Specifically, we introduce a sparse transformation layer atop the quantized dense embeddings, allowing the model to learn high-dimensional sparse representations that are both indexable and semantically aligned with the retrieval task.

6.2.1 Unified Model Architecture with Sparse Projection

Our model, denoted **`LSR_VQ`**, comprises three components: a frozen or trainable Contriever encoder, a vector quantization module, and a sparse projection layer inspired by KALE for indexable output. Given a tokenized input (query or passage), we compute its mean-pooled dense representation $\mathbf{h} \in R^d$ and apply chunked vector quantization as described in the above method. The resulting vector $\mathcal{Q}(\mathbf{h}) \in R^d$ is then passed through a sparse transformation layer:

$$\mathbf{s} = \log(1 + \text{ReLU}(W\mathcal{Q}(\mathbf{h}) + b)), \quad W \in R^{D\times d}$$

This transformation projects the embedding into a high-dimensional sparse space ($D \gg d$), suitable for symbolic retrieval via inverted index. To ablate the effectiveness of the VQ bottleneck, we also evaluate a variant that bypasses the VQ component by passing h directly to the sparse layer(LSR No VQ).

Retrieval is performed in two ways:

1. **Dot-product retrieval:** The dot product is computed between the final sparse vector representations after top-k sparsification.
2. **Query expansion (KALE-style):** After training the model and obtaining sparse representations of queries and passages, the latent terms are generated for each non-zero dimension in the sparse representation. These terms are appended to the original query or passage text, each with a constant term frequency. BM25 is then performed on the expanded text.

6.2.2 Training

We jointly train the encoder, vector quantizer and the sparse projection layer. We observed that training was highly unstable with randomly initialized VQ. To address this, we initialize the VQ module using the trained codebook obtained from **Method A**.

We train the model using a contrastive ranking objective over triplets of the form (query, positive passage, negative passages). Let $\mathbf{q}$, $\mathbf{d}^+$, and $\{\mathbf{d}_j^-\}$ denote the sparse representations of the query, its relevant passage, and a set of negatives, respectively. The contrastive loss is defined as:

$$\mathcal{L}_{\text{rank}} = -\log \frac{\exp(\text{sim}(\mathbf{q},\mathbf{d}^+))}{\exp(\text{sim}(\mathbf{q},\mathbf{d}^+))+\sum_j \exp(\text{sim}(\mathbf{q},\mathbf{d}_j^-))}$$

where $\text{sim}(\cdot,\cdot)$ denotes dot product similarity between sparse vectors.

When quantization is enabled, we additionally include a codebook reconstruction loss:

$$\mathcal{L}_{\text{rec}} = \sum_{i=1}^{C} \left\| \mathbf{h}^{(i)} - \mathcal{Q}(\mathbf{h}^{(i)}) \right\|^2$$

Codebook entries are updated using exponential moving averages (EMA). Straight-through estimator (STE) is used to allow gradients to flow through this non-differentiable quantization step during backpropagation.

To encourage sparsity and interpretability in the output space, we also apply FLOPS(Paria et al., 2020) regularization over the batch-averaged sparse vectors:

$$\mathcal{L}_{\text{sparse}} = \sum_{j\in V} \left(\frac{1}{N} \sum_{i=1}^{N} s_j^{(i)} \right)^2$$

where $s_j^{(i)}$ denotes the activation of the j-th dimension in the sparse vector $\mathbf{s}^{(i)} \in R^{|V|}$ for the i-th example in a batch of size N. The index set V corresponds to the sparse output space (e.g., projection vocabulary size). This loss penalizes dimensions that are consistently active across the batch, thereby encouraging sparsity and selectivity. We apply this regularization separately to queries and passages, using independent coefficients λ_q and λ_p.

Method	MRR@10	NDCG@10	R@10	R@1000
LSR (No VQ)	0.89	0.90	0.97	0.99
LSR_VQ	0.72	0.75	0.89	0.99
LSR_VQ + latent terms	0.89	0.91	0.97	0.99

Table 4: Results of Method B with $\sim$ 7.4K passages in the search space.

Method	MRR@10	NDCG@10	R@10	R@1000
LSR (No VQ)	0.46	0.52	0.69	0.98
LSR_VQ	0.23	0.27	0.42	0.91

Table 5: Results of Method B with $\sim$ 5L passages in the search space.

The final training objective is a weighted combination of all components:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{rank}} + \beta \cdot \mathcal{L}_{\text{rec}} + \lambda_q \cdot \mathcal{L}^{(q)}_{\text{sparse}} + \lambda_p \cdot \mathcal{L}^{(p)}_{\text{sparse}}$$

6.2.3 Experimentation

We perform controlled experiments to evaluate the impact of sparsity and quantization loss weighting on retrieval performance in Method B. Specifically, we apply FLOPs-based sparsity regularization on both query and passage sparse-projection vectors, with regularization coefficients $\lambda_q = \lambda_p = 0.5$. This setting was found to provide a strong balance between retrieval quality and sparsity, yielding compact yet effective representations. We also include the codebook reconstruction loss ($\mathcal{L}_{\text{rec}}$) throughout training, with its weight β set implicitly to 1.0. Together, these settings enable stable training, encourage meaningful codebook usage, and maintain strong top-k retrieval performance without sacrificing efficiency.

For training, we used a batch size of 128 and a learning rate of 1×10^{-4}. We train for 5 epochs using the AdamW optimizer with a linear warmup schedule. All models are trained using 16-bit mixed precision on a single NVIDIA A100 GPU. Evaluation is conducted on the MS MARCO development set using MRR@10, nDCG@10, and Recall@100 as primary metrics.

Due to computational constraints, we restrict the retrieval search space to only the pool of relevant documents for the queries (6,980) in the dev set; the results are reported in Table 4. This corresponds to a total of 7,433 passages. For better comparison, we also evaluate performance with an expanded search space consisting of 500,000 randomly sampled passages, as shown in Table 5. The comparison is made in two settings.

- **LSR (No VQ)** – The vector quantization module is omitted.
- **LSR_VQ** – The end-to-end approach described above, using a pretrained VQ (initialized using K-Means) initialization.

We set the final sparse layer projection dimension D to 8192, the latent vocabulary size. For top-k sparsification, we only retain 1024 passage terms and 256 query terms. A minimum-weight threshold of 1×10^{-5} is also further applied.

The final average sparsity results are presented in Table 6. We also evaluate performance under high-sparsity conditions by restricting the number of latent terms to 16 for queries and 64 for passages, similar to KALE. The results are reported in Table 7.

Method	Avg. Query Sparsity	Avg. Passage Sparsity
LSR (No VQ)	240	431
LSR_VQ	**38.7**	**69.3**

Table 6: Average number of non-zero latent vocabulary terms in query and passage representations after sparsification.

Method	MRR@10	NDCG@10	R@10	R@1000
LSR (No VQ)	0.39	0.42	**0.88**	0.91
LSR_VQ	**0.70**	**0.74**	**0.88**	**0.99**

Table 7: Comparison of results with a fixed number of latent vocabulary terms: 16 for queries and 64 for passages

7 Contributions of Group Members

The technical workload was divided equally amongst all the team members to ensure balanced involvement across all core components.

- **Harshitha & Sreenivas**: Implemented the vector quantization framework, including chunked embedding, symbolic token mapping, and codebook structure.
- **Durga & Rohith**: Built and trained the LSR-VQ model, integrated the sparse projection layer, and implemented the combined loss function involving contrastive, codebook, and sparsity objectives.
- **Harshitha & Sreenivas**: Developed the retrieval pipeline over latent token sequences using BM25, CSR, and inverted indices; ran evaluations across different cluster sizes, embedding dimensions, and chunk configurations.
- **Rohith & Durga**: Conducted experiments with Residual VQ, KMeans-based codebook initialization, and weighted query expansion; also handled dataset preparation, inference

utilities, and training reproducibility.

8 Conclusion

This project explored a novel approach to sparse retrieval by leveraging vector quantization (VQ) to learn symbolic representations in latent semantic space. Through the implementation of both a symbolic retrieval pipeline (Method A) and an end-to-end trainable sparse projection model (Method B, LSR-VQ), we aimed to combine the semantic strength of dense embeddings with the efficiency and interpretability of symbolic indexing.

One key takeaway was that discretizing embeddings via VQ can provide strong semantic signals even in purely symbolic retrieval, often rivaling dense retrieval baselines. However, fine-tuning the encoder in the end-to-end setting proved more challenging than expected, especially when balancing sparsity, codebook usage, and ranking quality. Achieving meaningful improvements required careful coordination between codebook learning, projection sparsity, and contrastive supervision.

We were particularly surprised by how competitive BM25 over latent tokens could be, and how sensitive the performance was to hyperparameters like chunk size and codebook dimensionality. Another nontrivial challenge was ensuring effective codebook initialization and utilization, especially under EMA updates.

This project also examined the use of vector quantization to construct efficient and effective sparse representations for retrieval. We initialized codebooks using K-means clustering, providing semantically grounded centroids that facilitate meaningful quantization. We explored both standard and Residual VQ, finding that deeper quantization (i.e., more quantizers) improves representation quality by capturing finer semantic detail. Residual VQ enables progressive refinement while keeping the representation compact.

One of the key outcomes of our method is the **high level of sparsity** achieved in the learned representations. This is likely due to the effect of vector quantization, which helps assign vectors to a more confined and discrete space, encouraging sparsity. Despite their compressed form, the VQ-generated codes alone are sufficiently expressive to support effective retrieval. Additionally, we applied query expansion using these sparse latent dimensions, similar to KALE(Borges et al., 2023), which further improved overall performance by enhancing both lexical coverage and semantic matching.

If we were to continue this work, we would evaluate our approaches on the entire corpus data, explore integrating learned residual quantization more deeply into the projection layer, experiment with differentiable codebook updates, and expand our evaluation to include zero-shot or cross-domain scenarios. Additionally, comparing more directly to established sparse coding models like SPLADE in a unified framework would provide a stronger empirical foundation.

Overall, this project offered valuable insight into how symbolic abstraction over latent embeddings can enable scalable and interpretable neural retrieval an exciting direction for future IR systems.

9 AI Disclosure

- Did you use any AI assistance to complete this proposal? If so, please also specify what AI you used.
  - Yes

If you answered yes to the above question, please complete the following as well:

- If you used a large language model to assist you, please paste *all* of the prompts that you used below. Add a separate bullet for each prompt, and specify which part of the proposal is associated with which prompt.
  - Is the grammar in ”[our text]” correct?
  - Check Grammar ”[our text]”
- **Free response:** For each section or paragraph for which you used assistance, describe your overall experience with the AI. How helpful was it? Did it just directly give you a good output, or did you have to edit it? Was its output ever obviously wrong or irrelevant? Did you use it to generate new text, check your own ideas, or rewrite text?
  - It was quite useful. It helped us correct our grammar and suggested new words to replace those that were sometimes unconsciously overused. Occasionally, it suggested new words when it wasn’t really necessary, but it was nice to have synonyms readily available. Yes, it did help us generate new text and rewrite a few words in certain scenarios.

References

Aggarwal, C. C. and Zhai, C. (2012). *A Survey of Text Clustering Algorithms*, pages 77–128. Springer US, Boston, MA.

Baevski, A., Zhou, H., Mohamed, A., and Auli, M. (2020). wav2vec 2.0: A framework for self-supervised learning of speech representations.

Bajaj, P., Campos, D., Craswell, N., Deng, L., Gao, J., Liu, X., Majumder, R., McNamara, A., Mitra, B., Nguyen, T., et al. (2016). Ms marco: A human generated machine reading comprehension dataset. *arXiv preprint arXiv:1611.09268*.

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022.

Borges, L., Martins, B., and Callan, J. (2023). Kale: Using a k-sparse projector for lexical expansion. In *Proceedings of the 2023 ACM SIGIR International Conference on Theory of Information Retrieval*, ICTIR '23, page 13–22, New York, NY, USA. Association for Computing Machinery.

Caron, M., Bojanowski, P., Joulin, A., and Douze, M. (2019). Deep clustering for unsupervised learning of visual features.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding.

Douze, M., Guzhva, A., Deng, C., Johnson, J., Szilvasy, G., Mazaré, P.-E., Lomeli, M., Hosseini, L., and Jégou, H. (2025). The faiss library.

Formal, T., Lassance, C., Piwowarski, B., and Clinchant, S. (2021a). SPLADE v2: Sparse lexical and expansion model for information retrieval. *CoRR*, abs/2109.10086.

Formal, T., Piwowarski, B., and Clinchant, S. (2021b). Splade: Sparse lexical and expansion model for first stage ranking.

Furnas, G. W., Landauer, T. K., Gomez, L. M., and Dumais, S. T. (1987). The vocabulary problem in human-system communication. *Commun. ACM*, 30(11):964–971.

Izacard, G., Caron, M., Hosseini, L., Riedel, S., Bojanowski, P., Joulin, A., and Grave, E. (2022). Unsupervised dense information retrieval with contrastive learning.

Karpukhin, V., Oğuz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., and tau Yih, W. (2020). Dense passage retrieval for open-domain question answering.

Khattab, O. and Zaharia, M. (2020). Colbert: Efficient and effective passage search via contextualized late interaction over bert.

Lei, Y., Ding, L., Cao, Y., Zan, C., Yates, A., and Tao, D. (2023). Unsupervised dense retrieval with relevance-aware contrastive pre-training. In Rogers, A., Boyd-Graber, J., and Okazaki, N., editors, *Findings of the Association for Computational Linguistics: ACL 2023*, pages 10932–10940, Toronto, Canada. Association for Computational Linguistics.

Lin, J. and Ma, X. (2021). A few brief notes on deepimpact, coil, and a conceptual framework for information retrieval techniques.

Malkov, Y. A. and Yashunin, D. A. (2018). Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs.

Mallia, A., Khattab, O., Tonellotto, N., and Suel, T. (2021). Learning passage impacts for inverted indexes.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space.

Paria, B., Yeh, C.-K., Yen, I. E. H., Xu, N., Ravikumar, P., and Póczos, B. (2020). Minimizing flops to learn efficient sparse representations.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Robertson, S. and Zaragoza, H. (2009). The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends in Information Retrieval*, 3:333–389.

van den Oord, A., Vinyals, O., and Kavukcuoglu, K. (2018). Neural discrete representation learning.

Xiao, S., Liu, Z., Shao, Y., and Cao, Z. (2022). Retromae: Pre-training retrieval-oriented language models via masked auto-encoder.

Xu, T. and Jiang, J. (2022). A graph adaptive density peaks clustering algorithm for automatic centroid selection and effective aggregation. *Expert Systems with Applications*, 195:116539.

Acknowledgments

We would like to sincerely thank Chris Samarinas for his invaluable support throughout the course of this project. His guidance in helping us explore various modeling directions, and set up the initial codebase was instrumental to our progress. We also acknowledge that approximately $77 was spent on Google Colab compute units to train, fine-tune, and evaluate our models.