ISSN: 2754-6659

Journal of Artificial Intelligence & Cloud Computing

SCIENTIFIC Research and Community

Review Article

Open Access

Efficient Context Retrieval in Large Language Models Using Low-Dimensional Embeddings

Naveen Koka

USA

ABSTRACT

Large Language Models (LLMs) have demonstrated remarkable capabilities in understanding and generating human language, but their deployment in enterprise scenarios often faces challenges related to **Cost, Scalability, and Efficiency**. Training or fine-tuning large models on customer-specific data is typically resource-intensive, while embedding vectors with thousands of dimensions can be expensive to store and slow to retrieve at scale.

This paper presents a **Lean Approach** to maximizing the value of LLMs with **Minimal Training and Low-Dimensional Embeddings**. We show how customer and field service data can be transformed into **Dense Vectors of Reduced Dimensionality** using unsupervised methods such as PCA or lightweight supervised projection heads trained on small labeled datasets. These compact embeddings (128–256 dimensions) are sufficient for high-quality semantic search, clustering, and classification while drastically reducing memory and compute requirements.

The proposed architecture combines **Vector Similarity Search (KNN, Cosine Similarity)** with **Retrieval-Augmented Generation** (RAG), enabling LLMs to deliver **context-Aware Responses** without costly fine-tuning. Further accuracy is achieved with **Lightweight Re-Ranking Models** (e.g., logistic regression) trained on limited examples. We illustrate this workflow in a **Field Service Use Case**, where the system retrieves and synthesizes knowledge from past service tickets and manuals to provide technicians with actionable troubleshooting guidance.

Our results highlight that **Most Practical Gains in Enterprise LLM Applications Come from Retrieval Quality and Representation Efficiency Rather than Full Model Retraining.** By focusing on compact embeddings and minimal additional training, organizations can achieve **Faster Inference, Lower Costs, and Robust Performance** in real-world customer support and field service scenarios.

***Corresponding author**

Naveen Koka, USA.

Received: August 29, 2025; **Accepted:** September 01, 2025; **Published:** September 11, 2025

Keywords: Mobile App Platforms, Offline, Data Sync

Introduction

Current In modern AI applications, understanding how LLMs interpret and retrieve context is crucial for both researchers and practitioners. For instance, in field service operations, an AI assistant may diagnose an equipment fault by recalling similar historical cases from a knowledge base.

This ability relies on three fundamental processes:

- Converting text into **Dense Vector Embeddings** that capture semantic meaning.
- Using the **Q–K–V Attention Mechanism** to relate different tokens in the input.
- Applying **SoftMax** to turn raw similarity scores into interpretable attention weights.

This paper demystifies these processes, using clear explanations, diagrams, and a practical example from the field service domain.

From Words to Dense Vectors

LLMs cannot directly process raw text; instead, they convert text into **dense vectors**—numerical representations in a high-dimensional space.

Tokenization

The first step is to break the input text into **Tokens** using a tokenizer such as Byte Pair Encoding (BPE) or Word Piece.

Example:
"Compressor Making Loud Noise"
Tokenized Output
["Compressor", " Making", " Loud", " Noise"]
Each token is assigned a unique **token ID**.

The **Embedding layer** is a lookup table—called the **Embedding Matrix**—with:

- **Rows** = Vocabulary Size (e.g., 50,000 tokens)
- **Columns** = Embedding Dimension (e.g., 768, 1024, 1536)

If "Compressor" has token ID 47738, row 47738 in the embedding matrix contains its **Dense Vector**, e.g.:
[0.12, -0.45, 0.88, 0.31, ...] # length = Embedding Dimension.

These vectors are **Learned During Training** so that semantically similar words have similar vectors.

The Q–K–V Attention Mechanism

Once we have embeddings for each token, the model needs to determine **Which Tokens Should Influence Each Other** when producing contextualized representations.

Generating Q, K, V
For Each Token Embedding X:
Q = xWQ, K = xWk, V = xWv
where WQ, Wk, Wx are learned parameter matrices.

- **Q (Query):** What this token is looking for in other tokens.
- **K (Key):** The searchable "tag" of this token.
- **V (Value):** The information this token will contribute if attended to.

Computing Similarity Scores
To determine how much one token should attend to another, we compute the dot product between the query of the current token and the keys of all tokens:

Score (Qi .Kj) = Qi . Kj

These **Raw Scores** (Logits) can be Positive or Negative and do not yet sum to 1.

Applying SoftMax
Given a vector of Raw Scores z = [z_1, z_2, ..., z_n]:
SoftMax (z_i) = e^(z_i) / Σ e^(z_j)

Where e^(z_i) exponentiates each score, ensuring all values are positive, and division by the sum normalizes the values so they add up to 1.

Weighted Sum of Values
The attention weights from SoftMax are used to compute a **Weighted Sum** of the value vectors VVV, producing a **Contextualized Embedding** for the token.

SoftMax in Attention
The **SoftMax Function** is a crucial component in the self-attention mechanism of LLMs. After computing the raw similarity scores (logits) between a token's **Query** and all tokens' **Keys**, SoftMax transforms these scores into a probability distribution that determines how much weight to give each token's **value** vector.

The SoftMax Formula
Given a vector of raw scores $Z=[z_1,z_2,z_3,....,z_n]$

$$softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

Where:

- e^{zi} exponentiates each score, ensuring all values are positive.
- Division by the sum normalizes the values so they add up to 1.
- The result is a **probability distribution** over all tokens.

Example: "Pump Leak Detected"
Suppose the token 'pump' has the following raw attention scores (logits) when compared to all tokens in the sequence:

Pump	2.0
Leak	1.0
Detected	0.1

After SoftMax, these become Approximately:

Pump	0.659
Leak	0.242
Detected	0.099

Context Building in Self-Attention
Self-attention doesn't just compute attention for one token — it does it for every token in the sequence simultaneously. This produces an attention matrix that captures the relationships between all pairs of tokens [1].

Attention for a Single Token
From Section 4, we saw that for "Pump", the model computes:

Token	Raw Score (Logit)	SoftMax Weight
Pump	2	0.659
Leak	1	0.242
Detected	0.1	0.099

This gives the attention distribution for "PUMP" over all tokens.

Attention for All Tokens
We repeat this process for each token:
"Leak" → compare its query Qleak with all keys
"Detected" → compare its query Qleak with all keys

Query \ Key	Pump	Leak	detected
Pump	0.659	0.242	0.099
Leak	0.301	0.524	0.175
Detected	0.210	0.370	0.420

Interpretation

- Row = "Who is Paying Attention"
- Column = "Who They are Attending to"
- Values = Probability Weights (after SoftMax)

$$\text{Context}_i = \sum_{j=1}^{n} \text{AttentionWeight}_{i,j} \cdot V_j$$

This produces a **Contextualized Embedding** for each token:

- "Pump"'s new vector incorporates weighted information from "Leak" and "detected".
- "Leak"'s new vector incorporates context from "pump" and "Detected", etc.

Building Context Across the Sequence
By the end of the self-attention calculation:

- Each token's representation is enriched with information from all other tokens.
- This allows the LLM to understand "pump leak detected" as a single event, not just three unrelated words.

Sentence-Level Embeddings and Context Retrieval
While self-attention enriches each token with contextual information, many real-world applications require a **single vector** to represent an entire sentence, paragraph, or document. This single vector — the **Sentence Embedding** — is used for

Semantic Search, Retrieval-Augmented Generation (RAG), and **Classification.**

From Token Embeddings to Sentence Embedding
There are several strategies to obtain a sentence embedding from contextualized token embeddings:

- **CLS Token Representation**
  In models like BERT, a special [CLS] token is prepended to the input.
  After the Transformer layers, the [CLS] vector is used as the sentence embedding.

- **Mean Pooling**
  Average all contextualized token vectors to form a single embedding.
- **Max Pooling**
  Take the maximum value across each dimension from all token vectors.

Example—Mean-Pooling:
If "pump leak detected" produces contextualized token vectors:

$$t1,t2,t3 \in Rd$$

Then the sentence embedding is:

$$S = \frac{t_1 + t_2 + t_3}{3}$$

Where d is the embedding dimension (e.g., 768).

Storing Embeddings in a Vector Database
Once sentence embeddings are generated for all documents or cases:

- Store them in a vector database such as Pinecone, Weaviate, Milvus, or PostgreSQL with pgvector.
- Each stored vector is linked to metadata (e.g., case ID, description, resolution).

Vector DB indexing uses algorithms like:

- **HNSW** (Hierarchical Navigable Small World) graphs.
- **IVF** (Inverted File Index).

Retrieval Using Cosine Similarity and KNN
When a new query arrives:

- Tokenize → embed → contextualize → pool → get query embedding.
- Compute similarity between query embedding and stored embeddings.
- Use **cosine similarity** or **KNN search** to find top-N most similar vectors.

Cosine similarity formula:

$$\text{sim}(a, b) = \frac{a.b}{||a||||b||}$$

Feeding Retrieved Context Back to the LLM

- Once Relevant Documents are Retrieved:
- Concatenate Them with the User's Query.
- Feed them into the LLM as Extended Context.
- The LLM Processes the Combined Text with Attention, Enabling Context-Aware Answers.

Field Service Example

- **Query:** "Low Pressure in Pump Model X After Seal Replacement".
- **Retrieval:** Embedding search Finds Similar Past Cases Involving Seal Leaks and Pump Performance Drops.
- **Generation:** LLM uses those Cases as Context to Produce a Troubleshooting Guide.

Solution
Dimensionality Reduction for Efficient Training
We propose using **Low-Dimensional Embeddings** that preserve semantic similarity while reducing computational overhead. Techniques like **Principal Component Analysis** (PCA), **Singular Value Decomposition (SVD)**, or **Low-Rank Matrix Factorization** can compress high-dimensional vectors (e.g., 768 or 1024) into a smaller latent space (e.g., 64–128) without significant loss of information.

This ensures:

- Lower training cost.
- Reduced memory footprint.
- Faster convergence during fine-tuning.

Context Preservation via Attention
The **Q–K–V Mechanism** ensures that even in reduced dimensions, the model captures relationships across tokens:

- Queries (Q) represent the current focus.
- Keys (K) represent possible matches.
- Values (V) carry contextual meaning.

The **SoftMax-normalized dot product of Q and K** ensures that relevant tokens still dominate, even if the embedding size is reduced [2-5].

Minimal Training with Pretrained Foundations
Instead of training from scratch, we fine-tune a **Pretrained LLM** using domain-specific embeddings:

- Start with embeddings from reduced-dimension vectors.
- Feed them into the pretrained transformer layers.
- Use lightweight fine-tuning methods like **LORA (Low-Rank Adaptation)** or **Prefix-Tuning**.
  This minimizes the amount of data and compute required.

Knowledge Injection through Embeddings
Customer data is stored as compressed embeddings in a Vector Database. At inference time:

- Retrieve Top-K Nearest Vectors for a Query.
- Pass Them Through the Attention Mechanism to Build a Contextualized Representation.
- Feed this Representation into the LLM For Generation.

This approach ensures that the LLM leverages customer data without retraining from scratch, giving **Maximum Benefit from Minimal Fine-Tuning** [6,7].

Conclusion
This work demonstrates that meaningful language understanding can be achieved with **Minimal Dimensionality and Training Overhead** by leveraging tokenization, attention-based contextualization, and efficient embedding compression. By reducing embedding dimensions while preserving semantic richness, and coupling this with lightweight fine-tuning methods, we enable large language models to adapt effectively to domain-specific customer data without the need for extensive retraining. The integration of vector-based retrieval further enhances

contextual grounding, ensuring that the model remains both computationally efficient and highly effective. Ultimately, this approach highlights a pathway toward **Scalable, Cost-Efficient, and Knowledge-Rich LLM Applications** that maximize benefit from minimal resources.

References

1. Bowen Peng, Jeffrey Quesnelle, Honglu Fan, Enrico Shippole (2023) YaRN: Efficient Context Window Extension of Large Language Models. 10.48550/arXiv.2309.00071.
2. Gongbo Zhang, Zihan Xu, Qiao Jin, Fangyi Chen, Yilu Fang, et al. (2025) Leveraging long context in retrieval augmented language models for medical question answering. npj Digit. Med 8: 239.
3. Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun Sophia Shao, et al. (2024) KVQuant: Towards 10 Million Context Length LLM Inference with KV Cache Quantization. https://arxiv.org/abs/2401.18079
4. Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, et al. (2024) Improving Text Embeddings with Large Language Models 11897-11916.
5. Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, et al. (2024) Retrieval-Augmented Generation for Large Language Models: A Survey. http://arxiv.org/abs/2312.10997
6. Alina Petukhova, João P Matos-Carvalho, Nuno Fachada (2025) Text clustering with large language model embeddings, International Journal of Cognitive Computing in Engineering 6: 100-108.
7. Choi Y, Chiu CY, Sontag D (2016) Learning Low-Dimensional Representations of Medical Concepts. AMIA Jt Summits Transl Sci Proc 2016: 41-50.