String inclusion for transformer embeddings

Bennett Bullock

February 2026

1 Introduction

Vector embeddings from transformers such as BERT or OpenAI are robust in representing contextual, lexical, and syntactic content of two strings. For strings of comparable length, cosine similarity of two embeddings is very reliable for determining whether the strings share this content. However, for strings of unequal length, cosine distance can be unreliable. If a shorter string, s_s, is contained in a longer string, s_l, a cosine similarity for their embeddings may be as low as 0.4. This is a problem for any use case requiring vector indices, such as RAG, as well as zero-shot classification using only embeddings, such as topic classification and entity recognition. Industry has found workarounds. For example, one solution is to include s_s and s_l in a prompt, and ask the LLM if the term is included. Another is to chop a sentence into n-grams, and take embeddings of those n-grams. Another is to concatenate a bag-of-words vector onto the embedding and see if a token sequence is in the bag-of-words vector. As one can imagine, these can very slow, computationally expensive, and in the last case, ineffective when there is not an exact string match between s_l and s_s.

In this paper, we will present an approach to string inclusion which requires only the embeddings and token lengths for s_s and s_l during runtime. We define "string" as a sequence of tokens, and "substring" as a subsequence of tokens. "String inclusion" of s_s in s_l occurs when s_l has a substring whose lexical, contextual, and syntactic content, called "content", is similar to s_s. We define $\vec{v}_l$ and $\vec{v}_s$ as embeddings of s_l and s_s using a transformer architecture such as BERT or OpenAI, and $\vec{x}_l$ and $\vec{x}_s$ as bag-of-words embeddings such as frequency count, tf-idf, or one-hot. N_l and N_s are the number of tokens in s_l and s_s.

The solution will be based on an analogy to data compression. In this analogy, when s_s is included in s_l, we imagine that it is decompressed into s_l. Content in s_l that is generated by substrings of s_s is "signal", and content which is not generated by substrings in s_s is a compression artifact, or "noise". In this analogy, vector search is a search for the decompression of s_s with the least amount of noise. We outline our solution:

1. Assuming that our compression is lossy, the compression ratio can be expressed as PQ, where P describes noise and Q describes signal. This can then be expressed in terms of a value α.
2. α can apply to cosine similarity of bag-of-words embeddings, and transformer embeddings by extension.
3. The ranking metric, based on α and the compression ratio, has several appealing properties - negative rank when $N_s > N_l$, meaningful comparability of s_l's of highly different token counts, and high likelihood of string inclusion when the ranking metric is above 1 and certain other coditions are met.

2 String inclusion as compression

Lossy and lossless compression. Compression is a useful analogy because it includes token counts, and because it can be defined as lossy or lossless. In lossless compression, a compression function c has a decompression function d where $d(c(s_l)) = s_l$. In lossy compression, this is not the case - s_l contains noise.

A compression algorithm is evaluated by a compression ratio $\frac{N_l}{N_s}$. Although N_l and N_s are average token counts when we describe a compression algorithm, here we apply it to specific pairs of s_s and s_l. A high compression ratio means that a shorter s_s is required to decompress to the same s_l. It is important to remember that this is only a measure of quality with lossless compression. With lossy compression, it only compares token counts.

Factorizing the compression ratio. To relate the compression ratio to lossiness, we decompose the compression ratio using terms: $P = \frac{N_p}{d_p}$, where N_p is the token count for signal, and $Q = \frac{N_q}{d_q}$, where N_q is a count for noisy tokens. d_p and d_q are undefined denominators. We can express the compression ratio additively, as $P + Q$, or multiplicatively, as PQ. In the additive defintion, there is no relationship between changes in N_q and N_p. We know that with transformer embeddings, tokens are interrelated according to context and syntax, such that an increase in N_q will often lead to an increase in N_p. Therefore, we choose the multiplicative definition, defining P and Q in terms of a value $0 \leq \alpha \leq 0$:

$$\frac{N_l}{N_s} = PQ = \left(\frac{N_l}{N_s}\right)^{1-\alpha} \left(\frac{N_l}{N_s}\right)^{\alpha} \tag{1}$$

We assume that α determines the lossiness of the compression, so $\alpha = 1$ indicates lossless compression, and $\alpha = 0$ indicates compression so lossy that all substrings in s_l are noise. Our ranking will be the right factor.

3 Relating α to cosine similarity of embeddings

α with bag-of-words embeddings. We must define α in terms of the embeddings $\vec{v_l}$ and $\vec{v_s}$. We will do this indirectly, by first defining it in terms of a hypothetical bag-of-words feature space of n dimensions, where each dimension is associated with a substring of s_l and s_s. In this feature space, the embeddings are L2-normalized vectors $\vec{x}_l$ and $\vec{x}_s$. We then posit a norm-preserving matrix $P \in \mathbb{R}^{n \times n}$, which performs the mapping $P\vec{x}_s$. We remember that P is not necessarily a generalized mapping. It can exist for just one pair of s_s and s_l. In lossless compression, $\vec{x}_l = P\vec{x}_s$, and, because of L2-normalization:

$$||P\vec{x}_s|| = \vec{x}_l^T P\vec{x}_s = 1 \tag{2}$$

All substrings of all nonzero elements in s_l are signal, since they are mappings of substrings of s_s. With lossy compression, $\vec{x}_l^T P\vec{x}_s$ decreases. Since this value is between 0 and 1, it is a candidate for α, although explicitly building this feature space is not feasible.

α with transformer embeddings. To relate α to transformer embeddings, we factorize P into $A \in \mathbb{R}^{n \times m}$ and $B \in \mathbb{R}^{m \times n}$, where $m < n$, A and B are norm-preserving, and A has an inverse. We then restate the similarity above in terms of m-dimensional vectors:

$$\vec{x}_l^T P\vec{x}_s = \vec{x}_l^T AB\vec{x}_s = \left(A^{-1}\vec{x}_l\right)^T (B\vec{x}_s) \tag{3}$$

Because the right-hand side contains L2-normalized vectors, this is a cosine similarity in a lower-dimensional space. Because we can have different P's for individual pairs of s_l and s_s, we do not have to make any assumptions about how effective P is at dimension reductionality, and we can easily assume that this is a cosine similarity of transformer embeddings. $\vec{v}_l^T \vec{v}_s$ is now an estimation of α.

4 A compression-based ranking

With this definition, a ranking metric for s_l and s_s is:

$$\left(\frac{N_l}{N_s}\right)^{\vec{v}_l^T \vec{v}_s} \tag{4}$$

We apply a logarithm:

$$\vec{v}_l^T \vec{v}_s \left[\mathbf{log}(N_l) - \mathbf{log}(N_s)\right] \tag{5}$$

We define $\mathbf{sim}_{s,l} = \vec{v}_l^T \vec{v}_s$, and $\Delta_{s,l} = \mathbf{log}(N_l) - \mathbf{log}(N_s)$:

$$\mathbf{sim}_{s,l}\Delta_{s,l} \tag{6}$$

This has several appealing properties. First, $\Delta_{s,l}$ is negative when s_s is longer than s_l, sending s_l to the bottom of the ranking. Secondly, because $\mathbf{log}(N_l)$

grows according to a power law, two very long s_l's can be compared to s_s meaningfully even when the two N_l's are of unequal length. If the logarithm were base-10, two strings would only need to be in the same order of magnitude for a meaningful comparison. Thirdly, when the ranking is above 1 there is a surprising result:

$$1 \leq \mathbf{sim}_{s,l}\Delta_{s,l} \tag{7}$$

We can then define lower and upper bounds for $\mathbf{sim}_{s,l}$.

$$\frac{1}{\Delta_{s,l}} \leq \mathbf{sim}_{s,l} \leq 1 \tag{8}$$

We first see that if $\Delta_{s,l} = 1$, $\mathbf{sim}_{s,l}$ is 1, and therefore indicative of string inclusion. For lower values of $\mathbf{sim}_{s,l}$, (8) describes two intervals, $i_0 = \left[\frac{1}{\Delta_{s,l}}, \mathbf{sim}_{s,l}\right]$ and $i_1 = [\mathbf{sim}_{s,l}, 1]$, with sizes $d_0 = \left|\mathbf{sim}_{s,l} - \frac{1}{\Delta_{s,l}}\right|$ and $d_1 = |1 - \mathbf{sim}_{s,l}|$. We then have several interpretations:

1. $d_1 = 0$ indicates string inclusion with a probability of 1.
2. $d_0 = 0$ indicates lowest probability of string inclusion.
3. $d_0 > d_1$ indicates higher probability of string inclusion.
4. $d_0 < d_1$ indicates lower probability of string inclusion.

Therefore, we can sketch out an algorithm to determine string inclusion:

1. Apply ranking from (6) to all s_l, storing each $\Delta_{s,l}$ and $\mathbf{sim}_{s,l}$.
2. Select all rankings of above 1.
3. Compute d_0 and d_1 for each selection.
4. For each pair d_0, d_1:
   (a) If $d_0 = 0$, label pair "strong string inclusion".
   (b) If $d_0 > d_1$, label pair "probable string inclusion".
   (c) If $d_1 > d_0$, label pair "less probable string inclusion".
5. Divide items from 4 into three bins based on their labels. Preserve the original ranking from (6) within these bins.

5 Conclusion

In this short paper, we have derived an algorithm to determine string inclusion using only token count and cosine similarity of embeddings. Drawing an analogy to compression, we modeled vector search as a decompression problem, which led to geometric factoring of the compression ratio according to an exponent α.

Then, we relate α to cosine similarity of transformer embeddings by deriving these embeddings from bag-of-words embeddings. The resulting rank, in (6), has several appealing properties - being negative when s_l is shorter than s_s, scaling logarithmically to allow meaningful comparison of longer strings with unequal length, and giving a near-certain indication of string similarity when (6) is above 1.

I am hoping to incorporate this algorithm into a vector index, such that users will be able to try this ranking and hopefully enjoy the benefits.