# **Word Embeddings and Positional Encoding in Large Language Models: Foundations and Recent Advancements**

## **1. Introduction**

Large Language Models (LLMs) have demonstrated remarkable progress across a multitude of Natural Language Processing (NLP) tasks, showcasing an unprecedented ability to process, understand, and generate human-level text.1 This transformative impact is underpinned by fundamental representation techniques, most notably word embeddings and positional encoding, which serve as core components enabling LLMs to effectively interpret and produce coherent language.5 This report aims to provide a comprehensive overview of these essential techniques, encompassing their definitions, purposes, various methods of generation, their historical evolution within NLP and sequence modeling, and a detailed examination of recent advancements in the context of LLMs.

## **2. Understanding Word Embeddings in LLMs**

At their core, word embeddings in LLMs are vector representations of words, phrases, or even entire texts, meticulously generated by language models.5 The primary function of these embeddings is to capture the intricate semantic meaning, contextual nuances, and underlying relationships between linguistic units within a high-dimensional space.5 This capability allows LLMs to move beyond surface-level processing and achieve a deeper comprehension of language.

The development of word embeddings marked a significant departure from earlier, less sophisticated methods such as one-hot encoding.9 One-hot encoding represents each word as a sparse vector with a dimensionality equal to the size of the vocabulary, where only a single element corresponding to the word is 'hot' (usually represented as 1), and all other elements are zero. While this method allows for the differentiation of words, it fundamentally fails to capture any semantic or syntactic relationships between them.11 For instance, in a one-hot encoded space, the vectors for "king" and "queen" would be equidistant from each other, despite their semantic relatedness.

Word embeddings offer several key advantages over such sparse representations. Firstly, they provide a significant reduction in dimensionality.9 Instead of a vector whose size is the entire vocabulary (which can be hundreds of thousands or even millions for large corpora), word embeddings map words to dense vectors in a much lower-dimensional space, typically ranging from hundreds to thousands of dimensions.9 Secondly, these dense vectors are designed to represent semantic similarity.5 Words with similar meanings are positioned closer to each other in the embedding space, allowing the model to recognize and leverage these relationships. Furthermore, LLM embeddings can offer contextual awareness.9 Unlike traditional word embeddings, LLM-generated embeddings can capture different meanings of a word based on its surrounding words in a sentence. For example, the embedding for "bank" in "river bank" would be different from its embedding in "bank account".9 This contextuality is crucial for understanding the subtle nuances of language. Additionally, embeddings generated by a model trained on a large corpus can often be transferred and fine-tuned for various downstream NLP tasks, such as text classification, sentiment analysis, and machine translation, without the need for training from scratch.9 Finally, LLM embeddings exhibit robustness to linguistic phenomena such as polysemy (words with multiple meanings) and synonymy (different words with similar meanings).9

Before words can be transformed into embeddings, the raw text must first undergo a process called tokenization.6 Tokenization breaks down the input text into smaller units or tokens, which can be words, subwords, or even characters, depending on the desired granularity.6 These tokens then serve as the basic units that are mapped to their corresponding embedding vectors.11

## **3. Methods for Generating Word Embeddings**

Several techniques have been developed to generate word embeddings, playing a crucial role in the evolution of NLP and the capabilities of LLMs. Some of the key methods include Word2Vec, GloVe, and fastText.

### **3.1. Word2Vec**

Word2Vec, developed by Google in 2013, was a revolutionary technique that significantly advanced the field of NLP by providing an efficient way to learn high-quality word embeddings.10 It utilizes shallow, two-layer neural networks to learn these embeddings by modeling the linguistic contexts of words within a large corpus of text.21 The underlying principle of Word2Vec is the distributional hypothesis, which posits that words that appear in similar contexts are likely to have similar meanings.10

Word2Vec employs two main architectures for training word embeddings: Continuous Bag of Words (CBOW) and Skip-gram.6 The CBOW model aims to predict a target word based on the surrounding context words within a defined window.6 Given a sequence of words, CBOW uses the context (words before and after the target word) as input to a neural network to predict the target word.12 Conversely, the Skip-gram model takes a target word as input and tries to predict the surrounding context words.6 The skip-gram architecture typically weighs nearby context words more heavily than more distant ones.21

The training process for both CBOW and Skip-gram involves iterating over a large corpus of text. The models learn the word embeddings by adjusting the weights of the neural network to maximize the probability of correctly predicting the target word (in CBOW) or the context words (in Skip-gram).10 The resulting weights of the hidden layer in these shallow neural networks are then used as the word embeddings.10 Word2Vec's ability to capture semantic relationships, such as analogies (e.g., the vector difference between "king" and "man" is similar to the vector difference between "queen" and "woman"), demonstrated a significant step forward in machine understanding of language.12

### **3.2. GloVe (Global Vectors for Word Representation)**

GloVe, an acronym for Global Vectors for Word Representation, was developed at Stanford University and launched in 2014.20 This technique offers a different approach to learning word embeddings by directly leveraging the global co-occurrence statistics of words within a text corpus.17 GloVe combines the advantages of global matrix factorization methods, like Latent Semantic Analysis (LSA), with the local context window methods used in Word2Vec.27

The core idea behind GloVe is to construct a word-word co-occurrence matrix, where each element represents how often two distinct words appear together within a defined context window across the entire corpus.28 Unlike Word2Vec, which primarily focuses on local context during training, GloVe aims to capture both local and global semantic relationships by considering the frequency of word co-occurrences across the entire dataset.28 After constructing this co-occurrence matrix, GloVe employs matrix factorization techniques, such as Singular Value Decomposition (SVD), to extract the latent structure within the matrix and derive word vectors.28 The objective of the GloVe model is to learn two vectors for each word such that their dot product equals the logarithm of the words' probability of co-occurrence.27 This approach ensures that the distance between word vectors in the embedding space reflects their semantic relatedness.17 GloVe has shown strong performance on various NLP tasks, including word analogy and named entity recognition.10

### **3.3. fastText**

fastText, developed by Facebook, is an extension of the Word2Vec model that was introduced to address some of its limitations, particularly in handling out-of-vocabulary words and morphologically rich languages.6 The key innovation of fastText is its ability to exploit subword information by considering character n-grams within words.6 Instead of treating each word as an atomic unit, fastText represents words as the sum of the vector representations of its constituent character n-grams, along with the vector representation of the whole word itself.25 For example, the word "eating" might be represented by the n-grams "<ea", "eat", "ati", "tin", "ing", "ng>" and the word itself "<eating>".25

This subword approach allows fastText to generate embeddings for words that were not present in the training data.6 When an out-of-vocabulary word is encountered, fastText can construct its embedding by summing the embeddings of its character n-grams. This is a significant advantage over Word2Vec and GloVe, which typically assign a random or zero vector to such words.14 Furthermore, fastText performs particularly well for morphologically rich languages, where words can have many variations based on prefixes, suffixes, and inflections.25 Similar to Word2Vec, fastText utilizes both the Continuous Bag of Words (CBOW) and Skip-gram architectures for training word embeddings.25

### **3.4. Comparison of Word Embedding Techniques (Pre-LLM Era)**

The following table summarizes the key characteristics of the major word embedding techniques discussed above:

| **Technique** | **Year Introduced** | **Core Principle** | **Handling of Out-of-Vocabulary Words** | **Strengths** | **Weaknesses** |
| --- | --- | --- | --- | --- | --- |
| **Word2Vec** | 2013 | Distributional hypothesis: words in similar contexts have similar meanings | Assigns a random or zero vector | Efficient training, captures semantic relationships and analogies | Cannot handle out-of-vocabulary words effectively |
| **GloVe** | 2014 | Leverages global word co-occurrence statistics | Assigns a random or zero vector | Captures both local and global semantic relationships | Less effective for morphologically rich languages |
| **fastText** | 2016 | Exploits subword information (character n-grams) | Can generate embeddings for OOV words based on n-gram embeddings | Effective for OOV words and morphologically rich languages | Can be slower to train than Word2Vec |

This table provides a concise overview of the evolution of word embedding techniques before the advent of modern LLMs, highlighting their progression in addressing the challenges of representing word meaning numerically.

## **4. Positional Encoding in LLMs: The Necessity of Order**

Unlike Recurrent Neural Networks (RNNs), which process sequential data step by step, Transformer-based Large Language Models (LLMs) are designed to process all tokens within an input sequence in parallel using the self-attention mechanism.7 While this parallel processing offers significant computational advantages, the self-attention mechanism itself is inherently permutation-invariant.40 This means that if the order of words in a sentence is changed, the self-attention mechanism, without additional information, would treat the permuted sentence as having the same meaning as the original.40

However, the order of words in a sentence is crucial for conveying meaning. For instance, "The cat sat on the mat" has a completely different meaning from "The mat sat on the cat." To address this, LLMs employ a technique called positional encoding.7 Positional encoding is used to inject information about the position of each word within the sequence directly into the model.7 By adding positional information to the input embeddings, Transformers can maintain an awareness of word order, which is essential for correctly interpreting the structure and meaning of sentences.7 For example, positional encoding helps the model distinguish between "John loves Mary" and "Mary loves John" by providing unique positional information for each word, allowing the attention mechanism to consider the role of each word based on its position in the sentence.39

## **5. Techniques for Implementing Positional Encoding**

Two primary techniques are commonly used to implement positional encoding in LLMs: sinusoidal positional embeddings and learned positional embeddings.

### **5.1. Sinusoidal Positional Embeddings**

Sinusoidal positional embeddings were introduced in the original Transformer paper and remain a widely used method.47 This technique uses sine and cosine functions of different frequencies to encode the absolute position of each word in the input sequence.7 The positional encoding vector has the same dimensionality as the word embeddings. For each position in the sequence, a unique positional encoding vector is generated, and this vector is then added element-wise to the corresponding word embedding.7

The mathematical formulation for sinusoidal positional encoding typically involves using pairs of sine and cosine functions with varying frequencies. The frequency of these functions decreases along the dimensions of the embedding vector, ensuring that each position in the sequence is assigned a distinct encoding.38 One of the key advantages of sinusoidal positional embeddings is their ability to potentially generalize to sequence lengths longer than those encountered during training.51 Because the positional encodings are generated using a fixed mathematical function, the model can, in principle, compute encodings for arbitrarily long sequences. Furthermore, sinusoidal positional encoding allows the model to easily learn to attend by relative positions.38 For any fixed offset *k*, the positional encoding for position *p+k* can be represented as a linear function of the positional encoding for position *p*, which facilitates the learning of relationships between words at different relative distances.52

### **5.2. Learned Positional Embeddings**

Learned positional embeddings offer an alternative approach where the positional encodings are not determined by a fixed function but are instead treated as learnable parameters that are optimized during the model's training.38 In this method, each position in the input sequence up to a predefined maximum length is associated with a unique embedding vector. These positional embedding vectors are initialized randomly or with some predefined values and are then learned along with the other parameters of the model during the training process.48 Similar to sinusoidal embeddings, learned positional embeddings are added to the word embeddings before being fed into the Transformer layers.

The primary advantage of learned positional embeddings is their flexibility.38 The model can directly learn the optimal representation of position for the specific task and dataset it is being trained on, potentially capturing more nuanced positional information compared to the fixed sinusoidal approach. However, a key limitation of learned positional embeddings is their potential difficulty in generalizing to sequence lengths significantly longer than the maximum length seen during training.59 If the model has only been trained on sequences up to a certain length, it may not have learned appropriate positional embeddings for longer sequences encountered during inference. Models like BERT and the GPT series often utilize learned positional embeddings.59 Recent research has explored the structure of learned positional embeddings, suggesting that they might occupy a low-dimensional subspace within the embedding space.61

## **6. The Historical Evolution of Word Embeddings (Pre-LLM)**

The development of word embedding techniques in NLP before the advent of modern LLMs represents a significant evolution in how machines understand and process language.10 Early approaches in the field included the vector space model for information retrieval, where words were represented as vectors based on their occurrences in documents, leading to sparse and high-dimensional vectors that struggled to capture semantic relationships.15

In the late 1980s, Latent Semantic Analysis (LSA) was introduced as a method to reduce the dimensionality of these vectors and uncover latent semantic relationships by using techniques like Singular Value Decomposition (SVD) on the term-document matrix.15 Around the same time, the random indexing approach was developed for efficiently collecting word co-occurrence contexts.15 A foundational step towards modern word embeddings was taken by Yoshua Bengio and colleagues in the early 2000s (2000 and 2003), who introduced the concept of distributed representations of words, also known as word embeddings, trained jointly with the parameters of a neural language model.15 This work demonstrated the potential of neural networks to learn meaningful word representations.

In 2008, Collobert and Weston further highlighted the utility of pre-trained word embeddings by showing their effectiveness in improving performance on various downstream NLP tasks.33 However, it was the introduction of Word2Vec by Google in 2013 that truly popularized the use of word embeddings in NLP.10 Word2Vec's efficiency and ability to learn high-quality embeddings from large datasets led to its widespread adoption and spurred significant interest in the field. Following the success of Word2Vec, GloVe was developed at Stanford in 2014 as a competitive approach that leveraged global word co-occurrence statistics to learn word vectors.20 In 2016, fastText was introduced as an extension of Word2Vec that incorporated subword information, enabling better handling of out-of-vocabulary words and morphologically rich languages.6

The late 2010s saw the emergence of contextually-meaningful embeddings, such as ELMo and BERT.9 Unlike the static word embeddings produced by Word2Vec and GloVe (where each word has a single, fixed embedding), contextual embeddings are token-level, meaning that the embedding for a word occurrence can vary depending on the context in which it appears. This evolution from static to contextual embeddings marked another significant advancement in the ability of NLP models to understand the nuances of human language.

## **7. A Historical Perspective on Positional Encoding**

The ability to process sequential information has been a fundamental challenge in NLP. Before the advent of Transformer-based LLMs, Recurrent Neural Networks (RNNs), including their more sophisticated variants like Long Short-Term Memory networks (LSTMs) and Gated Recurrent Units (GRUs), were the dominant architectures for sequence modeling.20 RNNs inherently process sequences in order, maintaining a hidden state that is updated at each step based on the current input and the previous hidden state.7 This sequential processing naturally incorporates information about the position of words in the sequence.

However, RNNs faced limitations, particularly with long sequences, due to issues like the vanishing gradient problem, which made it difficult for them to learn long-range dependencies.42 Additionally, the sequential nature of RNNs hindered the parallelization of computation, making them less efficient for processing very large datasets.42

The Transformer model, introduced in 2017, offered a paradigm shift in sequence modeling by relying entirely on the self-attention mechanism to compute representations of the input and output without using sequence-aligned RNNs or convolution.7 The self-attention mechanism allows the model to attend to different parts of the input sequence when processing each token, enabling it to capture long-range dependencies more effectively and allowing for parallel computation.7 However, as discussed earlier, the self-attention mechanism by itself is order-agnostic. To compensate for this, the Transformer architecture incorporates positional encoding to explicitly provide the model with information about the position of each token in the sequence.7

The original Transformer paper explored both sinusoidal and learned positional embeddings.48 While both methods aimed to inject positional information into the model, the sinusoidal approach was favored in the initial architecture. Subsequently, many models, including BERT, adopted learned positional embeddings, highlighting the ongoing exploration and refinement of techniques for encoding sequential order in neural networks.48

## **8. Recent Breakthroughs in Word Embeddings for LLMs (2022-2025)**

Recent research in the field of NLP has increasingly focused on leveraging Large Language Models (LLMs) to generate word, sentence, and document embeddings for various downstream tasks.1 A key question in this area is whether the improved performance observed with these LLM-induced embeddings is simply a result of the scale of these models, or if the underlying embeddings they produce are fundamentally different from those generated by classical encoding models like Word2Vec, GloVe, Sentence-BERT (SBERT), or Universal Sentence Encoder (USE).1

Studies have shown that LLMs tend to cluster semantically related words more tightly and achieve better results on analogy tasks in decontextualized settings compared to traditional methods.1 This suggests that LLMs excel at capturing semantic relationships between words when considered in isolation. However, in contextualized settings, where the meaning of a word or sentence is influenced by its surrounding text, classical models like SimCSE have sometimes outperformed LLMs in sentence-level similarity assessment tasks.1 This highlights the continued relevance of these lighter, more specialized models for capturing fine-grained semantic nuances in context.

Another significant area of research involves fine-tuning open-source decoder-only LLMs on synthetic data generated by proprietary, larger LLMs.76 This approach has shown promising results in achieving strong performance on competitive text embedding benchmarks without relying on labeled data. Furthermore, combining synthetic and labeled data for fine-tuning has even led to new state-of-the-art results on established benchmarks like BEIR and MTEB.78 This suggests that knowledge distillation from powerful, albeit black-box, LLMs can be an effective strategy for enhancing the capabilities of smaller, more accessible models in generating high-quality text embeddings.

Researchers are also exploring architectural changes within LLMs to improve the handling of different types of input. For instance, the ASIDE (Architecturally Separated Instruction-Data Embeddings) architecture proposes using separate embedding layers for instructions and data within the model.79 This allows for a clearer distinction between these roles, potentially improving the model's ability to process and understand complex prompts.

The efficiency and adaptability of LLMs are also being addressed through research on vocabulary manipulation. Techniques for vocabulary pruning, which aims to reduce the size of the vocabulary without significantly impacting performance, and tokenizer extension, which involves adding new tokens to an existing tokenizer, are being investigated.80 These efforts can influence the quality and characteristics of the word embeddings generated by the model.

Furthermore, LLMs are being explored as powerful encoders in machine translation tasks.81 Recent findings suggest that using an LLM as an encoder, whose output is then fed into a more traditional NMT decoder, can achieve comparable or even better performance than baseline systems while offering significant improvements in inference speed and memory footprint.81 This indicates a potential direction for developing more efficient and effective machine translation systems by leveraging the strong language understanding capabilities of LLMs for encoding the source language.

Finally, ongoing research is dedicated to improving universal text embeddings, which aim to be general-purpose and perform well across a wide range of tasks and domains.76 This includes using multi-stage contrastive learning strategies with diverse training data, as well as focusing on developing improved loss functions that can better capture semantic similarity and address issues like gradient vanishing during training.76 Some of these approaches also leverage LLMs as data annotators to generate pseudo-supervised data for training, further highlighting the synergistic relationship between LLMs and the quest for better text embeddings.76

## **9. Latest Innovations in Positional Encoding for LLMs (2022-2025)**

Recent advancements in Large Language Models (LLMs) have also spurred significant research into improving positional encoding techniques, particularly focusing on addressing the challenges of handling long sequences and generalizing to unseen lengths.4 The impact of different positional encoding schemes on the ability of LLMs to generalize to longer sequences during inference is an active area of investigation.59 Some findings suggest that commonly used methods like ALiBi and Rotary might not be optimally suited for this task and can be outperformed by other techniques, such as T5's Relative PE, or even by Transformers that operate without any explicit positional encoding (NoPE) in certain scenarios.59

Relative positional embeddings, which encode the relative distance between tokens rather than their absolute position, are gaining prominence. Rotary Position Embedding (RoPE), used in models like Llama, is a notable example that employs a rotation matrix to encode positional information, offering flexibility in handling varying sequence lengths.48 The T5 model utilizes a technique called "relative bias," where a learnable scalar bias is used instead of an embedding vector for each relative position.48 Another approach involves integrating positional information into each transformer block of the model, rather than just adding it at the input layer, to improve the expressiveness of positional information throughout the network.48

To address the limitations of input length in LLMs, researchers are exploring prompt compression techniques.4 These methods aim to reduce the length of the input prompt while preserving the essential information needed for the LLM to perform the desired task. Some of these techniques involve manipulating the positional embeddings or the positions of tokens within the context.4 For example, techniques like soft prompting compress the context into dense memory slots by learning continuous representations in the latent space.4

Innovations like CacheFocus have been introduced to enhance length normalization and reduce inference latency in LLMs without requiring additional training.83 This method leverages query-independent, offline caching to reuse a Context Key-Value (KV) Cache Store efficiently. Additionally, an Adaptive Positional Allocation Strategy dynamically reassigns cache positions to maximize the use of the available positional encoding range, improving the model's ability to handle longer contexts, even exceeding the typical limits of models like LLaMA-2.83

Furthermore, there is research exploring alternative architectures to the dominant Transformer model. For instance, the RESONA framework proposes augmenting linear recurrent models with retrieval mechanisms to improve their in-context learning abilities.84 This approach could potentially offer a more computationally efficient way to handle long sequences compared to the quadratic complexity of self-attention in Transformers, potentially impacting the need for traditional positional encoding techniques.

## **10. Conclusion**

Word embeddings and positional encoding stand as foundational pillars in the architecture of Large Language Models, enabling them to achieve remarkable feats in understanding and generating human language. The evolution of word embeddings has progressed from sparse, high-dimensional representations to dense, context-aware vectors, significantly enhancing the ability of NLP models to capture semantic nuances. Techniques like Word2Vec, GloVe, and fastText laid the groundwork for the sophisticated embeddings used in modern LLMs. Similarly, the challenge of processing sequential data has been addressed through the development of positional encoding methods, which became crucial with the advent of the parallel processing Transformer architecture. Sinusoidal and learned positional embeddings provide LLMs with the necessary information about the order of tokens in a sequence.

Recent research continues to explore and refine these fundamental techniques. The field is actively investigating the benefits of using LLMs themselves to generate embeddings, while also recognizing the continued relevance of classical embedding models for specific tasks. A significant focus is on improving the ability of LLMs to handle longer sequences and generalize to unseen lengths, driving innovation in positional encoding techniques. Methods like relative positional embeddings, RoPE, and even the exploration of models without explicit positional encoding highlight the dynamic nature of this research area. Techniques like prompt compression and cache management further aim to enhance the efficiency and scalability of LLMs. As the demands on LLMs continue to grow, ongoing advancements in word embeddings and positional encoding will be crucial in pushing the boundaries of what these powerful language models can achieve.

#### Works cited

1. Revisiting Word Embeddings in the LLM Era - arXiv, accessed on April 11, 2025, <https://arxiv.org/html/2402.11094v3>
2. Word Embeddings Revisited: Do LLMs Offer Something New? - arXiv, accessed on April 11, 2025, <https://arxiv.org/html/2402.11094v2>
3. History, Development, and Principles of Large Language Models—An Introductory Survey, accessed on April 11, 2025, <https://arxiv.org/html/2402.06853v1>
4. arXiv:2503.19114v1 [cs.CL] 24 Mar 2025, accessed on April 11, 2025, <https://www.arxiv.org/pdf/2503.19114>
5. Embeddings 101: The Foundation of LLM Power and Innovation - Data Science Dojo, accessed on April 11, 2025, <https://datasciencedojo.com/blog/embeddings-and-llm/>
6. Understanding LLM Embeddings: A Comprehensive Guide - IrisAgent, accessed on April 11, 2025, <https://irisagent.com/blog/understanding-llm-embeddings-a-comprehensive-guide/>
7. History Of Transformer Models | Restackio, accessed on April 11, 2025, <https://www.restack.io/p/transformer-models-answer-history-of-transformers-cat-ai>
8. www.iguazio.com, accessed on April 11, 2025, <https://www.iguazio.com/glossary/llm-embeddings/#:~:text=LLM%20embeddings%20are%20vector%20representations,of%20the%20meaning%20of%20words.>
9. What are LLM Embeddings? - Iguazio, accessed on April 11, 2025, <https://www.iguazio.com/glossary/llm-embeddings/>
10. Word embeddings in NLP: A Complete Guide - Turing, accessed on April 11, 2025, <https://www.turing.com/kb/guide-on-word-embeddings-in-nlp>
11. What are LLM Embeddings? - Aisera, accessed on April 11, 2025, <https://aisera.com/blog/llm-embeddings/>
12. The Key to LLMs: A Mathematical Understanding of Word Embeddings - KDnuggets, accessed on April 11, 2025, <https://www.kdnuggets.com/the-key-to-llms-a-mathematical-understanding-of-word-embeddings>
13. What is Embedding? - Embeddings in Machine Learning Explained - AWS, accessed on April 11, 2025, <https://aws.amazon.com/what-is/embeddings-in-machine-learning/>
14. Word2vec: What Are Word Embeddings? A Complete Guide - Digitate, accessed on April 11, 2025, <https://digitate.com/blog/word2vec-what-are-word-embedding/>
15. Word embedding - Wikipedia, accessed on April 11, 2025, <https://en.wikipedia.org/wiki/Word_embedding>
16. 10 — Understanding Word2Vec 1: Word Embedding in NLP | by Aysel Aydin - Medium, accessed on April 11, 2025, <https://ayselaydin.medium.com/10-understanding-word2vec-1-word-embedding-in-nlp-90dc6e94250f>
17. Word Embedding using GloVe | Feature Extraction | NLP | Python - Hackers Realm, accessed on April 11, 2025, <https://www.hackersrealm.net/post/word-embedding-using-glove-python>
18. confused about embeddings and tokenization in LLMs : r/learnmachinelearning - Reddit, accessed on April 11, 2025, <https://www.reddit.com/r/learnmachinelearning/comments/1cs29kn/confused_about_embeddings_and_tokenization_in_llms/>
19. A Closer Look at Embeddings and Their Use in Large Language Models (LLMs), accessed on April 11, 2025, <https://kelvin.legal/embeddings_part2/>
20. A Brief History of Large Large Language Models (LLMs) - Idiot Developer, accessed on April 11, 2025, <https://idiotdeveloper.com/a-brief-history-of-large-large-language-models-llms/>
21. Word2vec - Wikipedia, accessed on April 11, 2025, <https://en.wikipedia.org/wiki/Word2vec>
22. Word Embeddings, accessed on April 11, 2025, <https://cbail.github.io/textasdata/word2vec/rmarkdown/word2vec.html>
23. NLP Illustrated, Part 3: Word2Vec - Towards Data Science, accessed on April 11, 2025, <https://towardsdatascience.com/nlp-illustrated-part-3-word2vec-5b2e12b6a63b/>
24. Word2Vec For Word Embeddings -A Beginner's Guide - Analytics Vidhya, accessed on April 11, 2025, <https://www.analyticsvidhya.com/blog/2021/07/word2vec-for-word-embeddings-a-beginners-guide/>
25. Introduction to FastText Embeddings and its Implication - - Analytics Vidhya, accessed on April 11, 2025, <https://www.analyticsvidhya.com/blog/2023/01/introduction-to-fasttext-embeddings-and-its-implication/>
26. Word representations - fastText, accessed on April 11, 2025, <https://fasttext.cc/docs/en/unsupervised-tutorial.html>
27. GloVe - Wikipedia, accessed on April 11, 2025, <https://en.wikipedia.org/wiki/GloVe>
28. GloVe Embeddings: 3 How To Python Tutorials & 9 Alternatives - Spot Intelligence, accessed on April 11, 2025, <https://spotintelligence.com/2023/11/27/glove-embedding/>
29. GloVe: Global Vectors for Word Representation, accessed on April 11, 2025, <https://nlp.stanford.edu/projects/glove/>
30. GloVe Word Embeddings - CRAN, accessed on April 11, 2025, <https://cran.r-project.org/web/packages/text2vec/vignettes/glove.html>
31. GloVe: Global Vectors for Word Representation - Stanford NLP Group, accessed on April 11, 2025, <https://nlp.stanford.edu/pubs/glove.pdf>
32. Master Embedding Techniques: Transform Your NLP Models Now - Data Science Dojo, accessed on April 11, 2025, <https://datasciencedojo.com/blog/embedding-techniques-and-language-models/>
33. On word embeddings - Part 1 - ruder.io, accessed on April 11, 2025, <https://www.ruder.io/word-embeddings-1/>
34. Embedding with FastText - dasarpAI, accessed on April 11, 2025, <https://dasarpai.com/dsblog/embedding-with-fasttext>
35. Word Embedding with FastText: A Deep Dive into Text Representations - YouTube, accessed on April 11, 2025, <https://www.youtube.com/watch?v=w8Fz6fzc1bs>
36. fastText Explained - Papers With Code, accessed on April 11, 2025, <https://paperswithcode.com/method/fasttext>
37. A Visual Guide to FastText Word Embeddings, accessed on April 11, 2025, <https://amitness.com/posts/fasttext-embeddings>
38. What is Positional Encoding? | Deepchecks, accessed on April 11, 2025, <https://www.deepchecks.com/glossary/positional-encoding/>
39. A Gentle Introduction to Positional Encoding in Transformer Models, Part 1 - MachineLearningMastery.com, accessed on April 11, 2025, <https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1/>
40. Is positional encoding necessary for transformer in language modeling? - Stack Overflow, accessed on April 11, 2025, <https://stackoverflow.com/questions/61440281/is-positional-encoding-necessary-for-transformer-in-language-modeling>
41. Why do we need Positional Encoding in Transformers? - YouTube, accessed on April 11, 2025, <https://www.youtube.com/watch?v=KJEhvJc9uW8>
42. Transformer (deep learning architecture) - Wikipedia, accessed on April 11, 2025, <https://en.wikipedia.org/wiki/Transformer_(deep_learning_architecture)>
43. How Transformers Work: A Detailed Exploration of Transformer Architecture - DataCamp, accessed on April 11, 2025, <https://www.datacamp.com/tutorial/how-transformers-work>
44. Position Embeddings for Vision Transformers, Explained - Towards Data Science, accessed on April 11, 2025, <https://towardsdatascience.com/position-embeddings-for-vision-transformers-explained-a6f9add341d5/>
45. www.deepchecks.com, accessed on April 11, 2025, <https://www.deepchecks.com/glossary/positional-encoding/#:~:text=Positional%20encoding%20is%20a%20technique,structure%20and%20meaning%20of%20sentences.>
46. An Overview of Large Language Models (LLMs) | ml-articles – Weights & Biases - Wandb, accessed on April 11, 2025, <https://wandb.ai/mostafaibrahim17/ml-articles/reports/An-Overview-of-Large-Language-Models-LLMs---VmlldzozODA3MzQz>
47. Understanding Transformer Neural Network Model in Deep Learning and NLP - Turing, accessed on April 11, 2025, <https://www.turing.com/kb/brief-introduction-to-transformers-and-their-power>
48. Building a Transformer LLM with Code: Evolution of Positional Encoding - Saurabh Yadav, accessed on April 11, 2025, <https://www.yadavsaurabh.com/building-a-transformer-llm-with-code-evolution-of-positional-encoding/>
49. Contextual Positional Encoding | Clio AI Research Insights, accessed on April 11, 2025, <https://www.clioapp.ai/research/cope>
50. Understanding positional encoding in Transformers | Oxford Protein Informatics Group, accessed on April 11, 2025, <https://www.blopig.com/blog/2023/10/understanding-positional-encoding-in-transformers/>
51. You could have designed state of the art positional encoding - Hugging Face, accessed on April 11, 2025, <https://huggingface.co/blog/designing-positional-encoding>
52. Transformer Architecture: The Positional Encoding - Amirhossein Kazemnejad's Blog, accessed on April 11, 2025, <https://kazemnejad.com/blog/transformer_architecture_positional_encoding/>
53. What is the positional encoding in the transformer model? - Data Science Stack Exchange, accessed on April 11, 2025, <https://datascience.stackexchange.com/questions/51065/what-is-the-positional-encoding-in-the-transformer-model>
54. How exactly does adding a positional encoding to a transformer's input embeddings model time-dependency? : r/LanguageTechnology - Reddit, accessed on April 11, 2025, <https://www.reddit.com/r/LanguageTechnology/comments/dxoyi6/how_exactly_does_adding_a_positional_encoding_to/>
55. sinusoidalPositionEncodingLayer - MathWorks, accessed on April 11, 2025, <https://www.mathworks.com/help/deeplearning/ref/nnet.cnn.layer.sinusoidalpositionencodinglayer.html>
56. Positional Encoding - Notes on AI, accessed on April 11, 2025, <https://notesonai.com/positional+encoding>
57. Positional encodings in transformers (NLP817 11.5) - YouTube, accessed on April 11, 2025, <https://m.youtube.com/watch?v=5V9gZcAd6cE>
58. [D] Relative positional embedding and what's the advantage over absolute positional encoding : r/MachineLearning - Reddit, accessed on April 11, 2025, <https://www.reddit.com/r/MachineLearning/comments/197euq9/d_relative_positional_embedding_and_whats_the/>
59. The Impact of Positional Encoding on Length Generalization in Transformers - arXiv, accessed on April 11, 2025, <https://arxiv.org/pdf/2305.19466>
60. why we use learnable positional encoding instead of Sinusoidal positional encoding, accessed on April 11, 2025, <https://ai.stackexchange.com/questions/45398/why-we-use-learnable-positional-encoding-instead-of-sinusoidal-positional-encodi>
61. Learned Transformer Position Embeddings Have a Low-Dimensional Structure - ACL Anthology, accessed on April 11, 2025, <https://aclanthology.org/2024.repl4nlp-1.17.pdf>
62. Why not \*train\* positional embeddings in Transformers? : r/MLQuestions - Reddit, accessed on April 11, 2025, <https://www.reddit.com/r/MLQuestions/comments/1bhkyos/why_not_train_positional_embeddings_in/>
63. [Discussion] Learned positional embeddings for longer sequences : r/MachineLearning, accessed on April 11, 2025, <https://www.reddit.com/r/MachineLearning/comments/1fb4zls/discussion_learned_positional_embeddings_for/>
64. The essence of learnable positional embedding? Does embedding improve outcomes better? - Stack Overflow, accessed on April 11, 2025, <https://stackoverflow.com/questions/73113261/the-essence-of-learnable-positional-embedding-does-embedding-improve-outcomes-b>
65. Why positional embeddings are implemented as just simple embeddings? - Hugging Face Forums, accessed on April 11, 2025, <https://discuss.huggingface.co/t/why-positional-embeddings-are-implemented-as-just-simple-embeddings/585>
66. Why BERT use learned positional embedding? - Cross Validated, accessed on April 11, 2025, <https://stats.stackexchange.com/questions/460161/why-bert-use-learned-positional-embedding>
67. Transformer Language Models without Positional Encodings Still Learn Positional Information - ACL Anthology, accessed on April 11, 2025, <https://aclanthology.org/2022.findings-emnlp.99.pdf>
68. 4 Stages of Word Embeddings: Making Machines Smarter - Data Science Dojo, accessed on April 11, 2025, <https://datasciencedojo.com/blog/evolution-of-word-embeddings/>
69. History and Future of LLMs | Exxact Blog, accessed on April 11, 2025, <https://www.exxactcorp.com/blog/deep-learrning/history-and-future-of-llms>
70. The history, timeline, and future of LLMs - Toloka, accessed on April 11, 2025, <https://toloka.ai/blog/history-of-llms/>
71. The History Of Natural Language Processing & Potential Future Breakthroughs [With Infographic Timeline] - Spot Intelligence, accessed on April 11, 2025, <https://spotintelligence.com/2023/06/23/history-natural-language-processing/>
72. FastText model — gensim - Radim Řehůřek, accessed on April 11, 2025, <https://radimrehurek.com/gensim/models/fasttext.html>
73. Word vectors for 157 languages - fastText, accessed on April 11, 2025, <https://fasttext.cc/docs/en/crawl-vectors.html>
74. History of Evolution of Language Models in NLP: Foundation of Generative AI and LLM, accessed on April 11, 2025, <https://www.youtube.com/watch?v=ohdvSwKaQDk>
75. [2502.19607] Revisiting Word Embeddings in the LLM Era - arXiv, accessed on April 11, 2025, <https://www.arxiv.org/abs/2502.19607>
76. Recent advances in universal text embeddings: A Comprehensive Review of Top-Performing Methods on the MTEB Benchmark - arXiv, accessed on April 11, 2025, <https://arxiv.org/html/2406.01607v2>
77. [2402.11094v2] Word Embeddings Revisited: Do LLMs Offer Something New? - arXiv, accessed on April 11, 2025, <https://arxiv.org/abs/2402.11094v2/>
78. Improving Text Embeddings with Large Language Models - arXiv, accessed on April 11, 2025, <https://arxiv.org/html/2401.00368v2>
79. arXiv:2503.10566v1 [cs.LG] 13 Mar 2025, accessed on April 11, 2025, <https://www.arxiv.org/pdf/2503.10566>
80. arXiv:2501.02631v1 [cs.CL] 5 Jan 2025, accessed on April 11, 2025, <https://arxiv.org/pdf/2501.02631>
81. arXiv:2503.06594v1 [cs.CL] 9 Mar 2025, accessed on April 11, 2025, <https://www.arxiv.org/pdf/2503.06594>
82. arXiv:2502.14409v1 [cs.CL] 20 Feb 2025, accessed on April 11, 2025, <https://www.arxiv.org/pdf/2502.14409>
83. arXiv:2502.11101v1 [cs.CL] 16 Feb 2025, accessed on April 11, 2025, <http://www.arxiv.org/pdf/2502.11101>
84. arXiv:2503.22913v1 [cs.CL] 28 Mar 2025, accessed on April 11, 2025, <https://arxiv.org/pdf/2503.22913>
85. The Impact of Positional Encoding on Length Generalization in Transformers - OpenReview, accessed on April 11, 2025, <https://openreview.net/forum?id=Drrl2gcjzl>
86. [2203.16634] Transformer Language Models without Positional Encodings Still Learn Positional Information - arXiv, accessed on April 11, 2025, <https://arxiv.org/abs/2203.16634>
87. Curriculum Vitae - Lingpeng Kong, accessed on April 11, 2025, <https://ikekonglp.github.io/lingpenk_cv.pdf>
88. Mohit Iyyer - Manning College of Information & Computer Sciences - UMass Amherst, accessed on April 11, 2025, <https://people.cs.umass.edu/~miyyer/data/cv.pdf>