THE TRANSFORMER ARCHITECTURE IN LARGE LANGUAGE MODELS

OLIVER MAJER

Fakulteta za matematiko in fiziko
Univerza v Ljubljani

The article presents the Transformer architecture, a foundational deep learning model that has driven the development of large language models. Central to this architecture is the attention mechanism, particularly the self-attention process, which allows the model to process input sequences of varying lengths and decode the semantic relationships among tokens. The article also explains how text is transformed into high-dimensional vector embeddings and how these representations contribute to understanding linguistic context. By delving into these key concepts, the article highlights the Transformer's influence on the development and efficiency of current state-of-the-art language models.

ARHITEKTURA TRANSFORMERJA V VELIKIH JEZIKOVNIH MODELIH

V članku je predstavljena arhitektura Transformer, temelj globokih nevronskih mrež, ki je spodbudil razvoj velikih jezikovnih modelov. Ključni element te arhitekture je mehanizem pozornosti, ki modelu omogoča obdelavo vhodov različnih dolžin in razčlenjevanje semantičnih razmerij med besedami. Pojasnjeno je tudi, kako se vhodno besedilo pretvori v visokodimenzionalen vektorje ter kako takšna predstavitev prispeva k razumevanju jezikovnega konteksta. Cilj članka je, da bralcu skozi poglabljanje v ključne koncepte Transformerja omogoči boljše razumevanje velikih jezikovnih modelov.

1. Introduction

Large language models (LLMs) have recently captured the public's attention and become ubiquitous in various domains. This ranges from conversational bots like ChatGPT to education and training systems. These models have proved efficient in understanding and generating human-like text [5]. However, the significant progress in the capabilities of LLMs has primarily been observed in recent years. A key influence in these advancements is the introduction of the Transformer architecture [2]. This article aims to present the core components of the Transformer architecture and consequently enable the reader to gain an understanding of how LLMs operate.

LLM is a computational model type, designed for natural language processing. It ranges from tasks such as understanding complex text passages to generating meaningful context-appropriate texts [5]. However, the development of LLMs faces three main challenges that need to be efficiently addressed in model design [3]:

- substantial variation in input length;
- capacity to process very large inputs;
- understanding and interpreting the complexity and ambiguity of the language, especially the long-range dependencies.

Consider the following passage:*"The companies are hesitant to hire new workforce, because they expect recession. It is foreseen to come in half a year."* In this scenario, the main objective for an LLM is to process the text in a manner that facilitates tasks such as predicting subsequent sentences or determining whether current conditions are favorable for the labor market. Additionally, the LLM must understand how the adjective "new" influences the meaning of the noun "workforce". Moreover, the pronoun references should be interpreted correctly, such as recognizing that the word "they" refers back to "The companies". Hence, grasping the semantics that shape the input's contextual meaning. The proposed challenges form the core motivation for the Transformer architecture discussed in this article. The transformer emerges as the solution to the three

problems. Understanding of how context influences the meaning of each word is tackled by *vector embedding*, discussed in Chapter 2, and *attention mechanism*, examined in Chapter 3. Additionally, the problems of variation in input length and capacity to process large inputs are addressed through the use of shared parameters across the model [3]. In the following sections, we will examine vector embeddings and components of attention mechanism in detail.

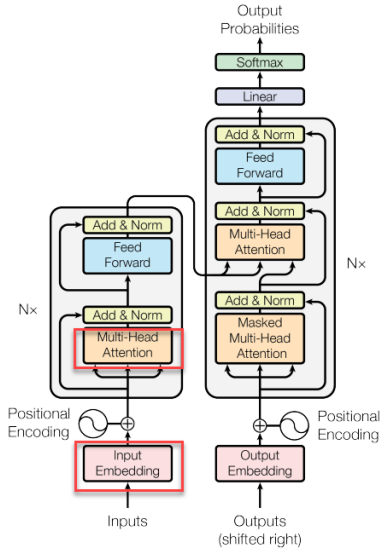

Figure 1. Overview of the Transformer model architecture. Vector embeddings and the attention mechanism will be further explored in detail in this article, highlighting their role in understanding context and handling complex input structures. Adapted from: [2]

2. Embedding Text into Vectors

For an LLM to process textual input, the text first has to be broken down into smaller units called tokens. A token can be a word, a word fragment, or even a punctuation mark. The exact form a token takes depends on the tokenization scheme used [6]. For the purpose of conceptual understanding, let us assume that each token is a word. For instance, the sentence "I love math" might be tokenized into "I", "love", and "math".

Subsequently, each token is mapped to a vector, which is essentially a list of real numbers. The numerical representation makes it more efficient for the computer to perform computations [7]. Formally, the numerical representation of the token is referred to as an *embedding vector*.

The embedding vector is designed to encode the semantic meaning of the token by representing it in a high-dimensional vector space. The core idea behind vector embedding is that tokens with similar meanings are located close to each other in this vector space. Each dimension in the embedding vector space can be thought of as encoding a specific idea or feature [8]. For instance, one direction might represent whether the token conveys a sense of time (e.g., past, future, or present) while another direction might encode formality (e.g., formal or colloquial expression). The value of the vector along a particular direction indicates the degree to which the token relates to the corresponding idea. This allows the model to effectively encode the token's semantic properties and its relationship to other tokens. For example, a well-trained embedding would position a token "football" closer to the token "ball" than to the token "lamp", reflecting their semantic similarity. The clustering in the embedding vector space ensures that tokens with similar meanings are located near each other [9].

Figure 2 provides a visual intuition of how word embeddings capture semantic relationships

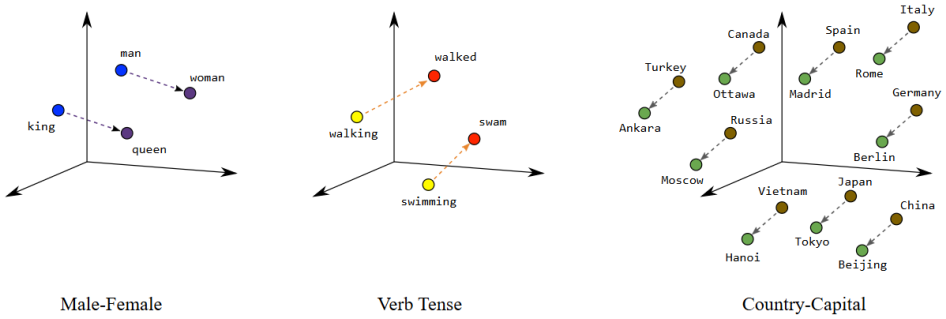

Figure 2. Word embeddings capture semantic relationships such as gender analogies, verb tenses, and country-capital pairs, illustrating semantic similarity in vector space. Adapted from: [1]

between tokens in the vector space. In the leftmost figure, the blue dot represents the token "man", and the purple dot represents the token "woman". The vector pointing from "man" to "woman" can be obtained by subtracting the embedding vector of "man" from that of "woman". Conversely, if the embedding vector of "king" is taken and the difference between the vectors "woman" and "man" is added to it, the resulting vector is found to be approximately located in the same position as the vector embedding for "queen". Mathematically, this relationship can be expressed by:

$$\overrightarrow{\text{queen}} \approx \overrightarrow{\text{king}} + (\overrightarrow{\text{woman}} - \overrightarrow{\text{man}})$$

This example demonstrates that specific directions in the embedding vector space encode meaningful relationships. In this case, the direction corresponds to the concept of gender. In the middle subfigure, a similar relationship is observed for the verbs, where the difference between two vectors encodes the transition from present tense to past tense. Whereas, the rightmost subfigure illustrates how the vector difference between country names and their capitals is consistent across multiple pairs.

3. Attention Mechanism

3.1 Motivation: The Need for Contextual Understanding

When the text is tokenized and each token is associated with an embedding vector, these vectors originally encapsulate only the general meaning of the token, independent of context. For example, consider the word "run" in the following three sentences:

- "I *run* every morning."
- "The coffee machine will *run* for 2 days."
- "She is *running* for president."

Humans intuitively understand that the word "run" has 3 significantly different meanings in the provided sentences depending on the context. However, the initial embedding of the token "run" remains fixed and encodes the general properties of the token, disregarding the specific context in the sentence. This raises an important question: How does the LLM architecture grasp the contextual meaning of a token within a sentence? The answer lies in the attention mechanism within the Transformer architecture.

3.2 Structure and Function of the Attention Mechanism

The attention mechanism allows the vectors to communicate with and influence one another. Hence, allowing the model to determine which words in a sentence are most relevant for updating the meaning of each token [2]. Essentially, the attention mechanism determines how much attention should each token allocate to other tokens. The process is reflected in modifications to the corresponding vectors in the embedding vector space.

Consider an attention mechanism that takes n input vectors, $\vec{x}_1, \ldots, \vec{x}_n$, where $\vec{x}_i$ is an embedding vector of size D for the i-th token in the input sequence. For each of the n input embedding vectors, the attention mechanism computes additional three vectors:

- Query vector $\vec{q}_i$
- Key vector $\vec{k}_i$
- Value vector $\vec{v}_i$

These vectors are obtained by multiplying the input embedding vector $\vec{x}_i$ with corresponding learnable weight matrices [2]. This process will be discussed in more detail in Chapter 4.

3.3 Query Vectors $\vec{q}_i$

Query vectors act as "questions" that tokens pose to determine their relationship with other tokens in the sequence. To understand this intuitively consider the following sentence: *"Pure mathematical objects are of great interest".* Take a closer look at the noun "objects". In this context, the token "objects" may query the sentence to identify the relevant adjectives that describe it, such as *"pure"* and *"mathematical"*. This conceptual "querying" is captured mathematically through the computation of the query vector, which will be discussed in Chapter 4.

3.4 Key Vectors $\vec{k}_i$

While query vectors $\vec{q}_i$ represent the "questions" embedding vector poses, key vectors $\vec{k}_i$ act as the "answers", indicating the degree of content relevance each key vector contains for a given query vector. Meaning that key vectors help determine how strongly tokens in the sequence relate to a particular query [3].

To deepen the understanding of the relationship between key and query vectors, let us revisit the sentence: *"Pure mathematical objects are of great interest".* When the token "objects" poses a query to identify its describing adjectives, tokens such as "Pure" and "mathematical" should provide relevant responses, encoded in their corresponding key vectors. This enables the attention mechanism to evaluate how closely the contents of the two adjectives align with the query posed by the token "objects". The computational aspects of this interaction will be explored further in Chapter 4.

3.5 Computing Attention Scores

Once the key vectors $\vec{k}_j$ and the query vectors $\vec{q}_i$ are obtained for all input tokens, the attention scores can be computed. These scores measure the semantic relevance between tokens in the input sequence and are calculated using the dot product [3]:

$$\vec{q}_i \cdot \vec{k}_j = \sum_{\ell=1}^{d} q_\ell \cdot k_\ell$$

where: q_ℓ and k_ℓ are the ℓ-th components of the query and key vectors, respectively.

Geometrically, the dot product measures how closely the query vector $\vec{q}_i$ and the key vector $\vec{k}_j$ are located in the shared query-key vector space. A larger positive dot product indicates that $\vec{k}_j$ and $\vec{q}_i$ are closely located in the vector space [10]. Hence, signifying a strong semantic connection between the corresponding tokens. On the contrary, if two tokens have completely different semantic meanings, the corresponding dot product will be negative or approaching zero, indicating minimal or no alignment.

Each dimension in the vector space can be thought of as encoding a specific idea or feature. The dot product quantifies how well these features align between the query and the key vector. Let a_{ij} represent the attention score, representing the interaction between token j and token i. The value a_{ij} indicates to what extent token j "pays attention" to token i based on semantic similarity and relevance.

3.6 Attention Pattern

Query / **Key**	Pure $\vec{q}_1$	mathematical $\vec{q}_2$	objects $\vec{q}_3$	are $\vec{q}_4$	of $\vec{q}_5$	great $\vec{q}_6$	interest $\vec{q}_7$
Pure $\vec{k}_1$	1.00	0.00	0.42	0.10	0.00	0.00	0.00
mathematical $\vec{k}_2$	0.00	1.00	0.58	0.10	0.00	0.00	0.00
objects $\vec{k}_3$	0.00	0.00	0.00	0.80	0.00	0.20	0.10
are $\vec{k}_4$	0.00	0.00	0.00	0.00	0.90	0.25	0.01
of $\vec{k}_5$	0.00	0.00	0.00	0.00	0.10	0.55	0.29
great $\vec{k}_6$	0.00	0.00	0.00	0.00	0.00	0.00	0.60
interest $\vec{k}_7$	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 1. Attention matrix illustrating the semantic relevance: the token "objects" poses a query, while the tokens "Pure" and "mathematical" respond as relevant adjectives. The rows highlighted in grey indicate the attention scores assigned by these two key tokens are the focus of analysis in this table. Note: the values are hypothetical and are included to illustrate the concept of attention score.

Table 1 provides a clear illustration of how the attention score reflects semantic relevance within a sentence. For instance, the key vectors $\vec{k}_1$ and $\vec{k}_2$ corresponding to the tokens "Pure" and "mathematical", respectively, evaluate their relevance to the query vector $\vec{q}_3$ ("objects"). The attention score a_{31} is 0.42, indicating that the key vector $\vec{k}_1$ assigns moderate attention to the key vector $\vec{q}_3$. Similarly, a_{32}, with value 0.58, demonstrates that $\vec{k}_2$ assigns even more attention to the query vector $\vec{q}_3$. These scores highlight the semantic relevance between the noun "objects" and its two descriptive adjectives "Pure" and "Mathematical". In contrast, tokens with minimal or no semantic relevance produce near-zero attention scores. For example, the attention score between $\vec{k}_1$ and other unrelated tokens, such as "of" is effectively 0.00. Hence, signifying their lack of relevance. The distinction in attention scores confirms the conceptual relationship between nouns and their corresponding adjectives.

Moreover, the grid shown in Table 1, often referred to as the *attention pattern*, visually presents the pairwise interaction between all query and key vectors within a sentence. The attention pattern

highlights a significant computational limitation: quadratic complexity. Specifically, the required space and computation of attention scores scale quadratically with the input sequence length n, resulting in a complexity of $O(n^2)$. This occurs because, for each of the n query vectors, attention scores must be computed for all n key vectors, producing an $n \times n$ attention grid [3].

Although the complexity grows quadratically, the attention mechanism remains independent of the input vector dimensionality D. This means that the complexity is determined by the number of input tokens, not the size of the query-key vector space. Consequently, the self-attention mechanism involves an inherent trade-off: semantic relevance between tokens can be captured across the entire input sequence, not just between adjacent tokens. However, this comes at the cost of quadratic growth in space and computational complexity.

3.7 The Softmax Function

In self-attention, the raw attention scores a_{ij} are computed as the dot products between the query vectors $\vec{q}_i$ and key vectors $\vec{k}_j$:

$$a_{ij} = \vec{k}_i \cdot \vec{q}_j.$$

The dot products self-attention mechanism produces raw attention scores, which can vary significantly, ranging from large values, much greater than 1, to negative values. This variability in attention scores presents a challenge in interpreting the scores as a measure of semantic relevance, as they do not form a probability distribution.

To convert these raw scores into probabilities known as *attention weights*, the Softmax function is applied along each row of the attention score matrix. The resulting attention weights α_{ij} represent how much the j-th key vector $\vec{q}_j$ "attends" to the i-th query vector $\vec{k}_i$.

To obtain the probability distribution from raw attention scores, the following requirements must be met:

- Each attention weight α_{ij} is an element of the interval $[0, 1]$;
- Attention weights in each row sum to 1.

The requirements are met by applying *Softmax* function to the raw dot products obtained in the attention mechanism. The Softmax function is a normalized exponential function that maps a set of n real numbers to a probability distribution over n possible outcomes. Each output lies in the range $[0, 1]$ and the sum of all outputs equals 1.

The Softmax function takes a vector of M real numbers as input and produces a probability distribution, where each component is normalized. For a vector $\vec{z} = (\vec{z}_1, \vec{z}_2, \ldots, \vec{z}_M) \in \mathbb{R}^M$, the Softmax function $\sigma(\vec{z})$ is defined as [4]:

$$\sigma : \mathbb{R}^M \to (0,1)^M, \quad \sigma(\vec{z})_i = \frac{e^{\vec{z}_i}}{\sum_{j=1}^{K} e^{\vec{z}_j}}, \quad i = 1, \ldots, M,$$

where $M > 1$ is the dimension of the input vector.

In the case presented in Table 1, the vector $\vec{z}$, which serves as the input to the softmax function, would have a dimensionality $M = 7$, corresponding to the number of key vectors for each query vector. Specifically, each column of the attention pattern contains 7 elements, each one corresponding to a key vector.

- Each component $\sigma(\vec{z})_i$ satisfies $0 < \sigma(\vec{z})_i < 1$, since $e^{\vec{z}_i} > 0$ for all real $\vec{z}_i$. The denominator $\sum_{j=1}^{M} e^{\vec{z}_j}$ is the sum of strictly positive terms and is strictly greater than $e^{\vec{z}_i}$. Mathematically,

$$\sum_{j=1}^{M} e^{\vec{z}_j} > e^{\vec{z}_i}.$$

Consequently,

$$\sigma(\vec{z})_i = \frac{e^{\vec{z}_i}}{\sum_{j=1}^{M} e^{\vec{z}_j}} < 1.$$

- The components of the output vector sum to 1:

$$\sum_{i=1}^{M} \sigma(\vec{z})_i = \sum_{i=1}^{M} \frac{e^{\vec{z}_i}}{\sum_{j=1}^{M} e^{\vec{z}_j}} = \frac{\sum_{i=1}^{M} e^{\vec{z}_i}}{\sum_{j=1}^{M} e^{\vec{z}_j}} = 1.$$

By applying the Softmax function, raw attention scores are transformed into a probability distribution. Hence, enabling the attention mechanism to compute a weighted sum of value vectors, where the weights α_{ij} reflect the relative importance of the key vectors to the query vector. Conceptually, key vectors "compete" with one another to contribute to the final result.

3.8 Value Vectors $\vec{v}_i$

Value vectors, computed for all n input tokens, encapsulate the semantic meaning of each token. Value vectors serve as containers of contextual information, dynamically adjusted based on attention scores. This adjustment mechanism is as follows:

1. All n value vectors $\vec{v}_i$ are scaled by the corresponding attention weights derived from the attention mechanism.
2. The scaled vectors are summed along the columns of the attention pattern.
3. The resulting vector, $\Delta\vec{v}_i$, is added to the original input vector $\vec{x}_i$ to produce an updated representation.

The result is a more refined vector, $\vec{x}_i + \Delta\vec{v}_i = \vec{x'}_i$, encoding the context to a greater extent. The described process is applied to all n embedding vectors. In Table 1, this can be thought of as performing a weighted sum of value vectors with the original input vector, applied iteratively for all n columns.

The process is referred to as a "single head of attention". It is parametrized by three shared parameter matrices: Query, Key, and Value matrix. This design ensures the model can process sequences of varying lengths without introducing additional parameters [3, 8].

4. Computation of Query-Key-Value Vectors

As discussed in Chapters 3.3 and 3.4 query vectors $\vec{q}_i$ act as "questions", and key vectors $\vec{k}_i$ as "answers", whereas value vectors $\vec{v}_i$ encapsulate the meaning of a token. All three vectors are computed by vector-matrix multiplication:

$$\vec{q}_i = \Omega_q \vec{x}_i$$

$$\vec{k}_i = \Omega_k \vec{x}_i$$

$$\vec{v}_i = \Omega_v \vec{x}_i$$

where query vector $\vec{q}_i$, key vector $\vec{k}_i$, or value vector $\vec{v}_i$ correspond to the i-th input vector. These vectors are computed through the vector-matrix multiplication of the input embedding vector $\vec{x}_i$ with learnable query weight matrix Ω_q, key weight matrix Ω_k, or value weight matrix Ω_v. The matrices remain constant across all n input vectors [3].

In addition, query vectors $\vec{q}_i$ and key vectors $\vec{k}_i$ can have a lower dimensionality than the input embedding vector, which notably reduces computational complexity. This reduction in dimensionality is achieved by applying the same matrix Ω_x across all n input vectors. Therefore, allowing the model to retain essential semantic information while minimizing resource usage.

Given that query and key vectors have the same dimensionality, they can be thought of as residing in the same vector space. This shared vector space enables the computation of the dot product, a central operation in the attention mechanism [3]. As previously discussed in Chapter 3.5 on the self-attention dot product, the shared dimensionality of query and key vectors is essential for computing their dot product efficiently.

The input text sequence is first divided into smaller fragments, referred to as tokens, which are assumed to be words for simplicity. Each token is then mapped to its corresponding embedding vector, which encodes the general semantic meaning of the token. These embedding vectors are subsequently processed through the attention mechanism as illustrated in Figure 3.

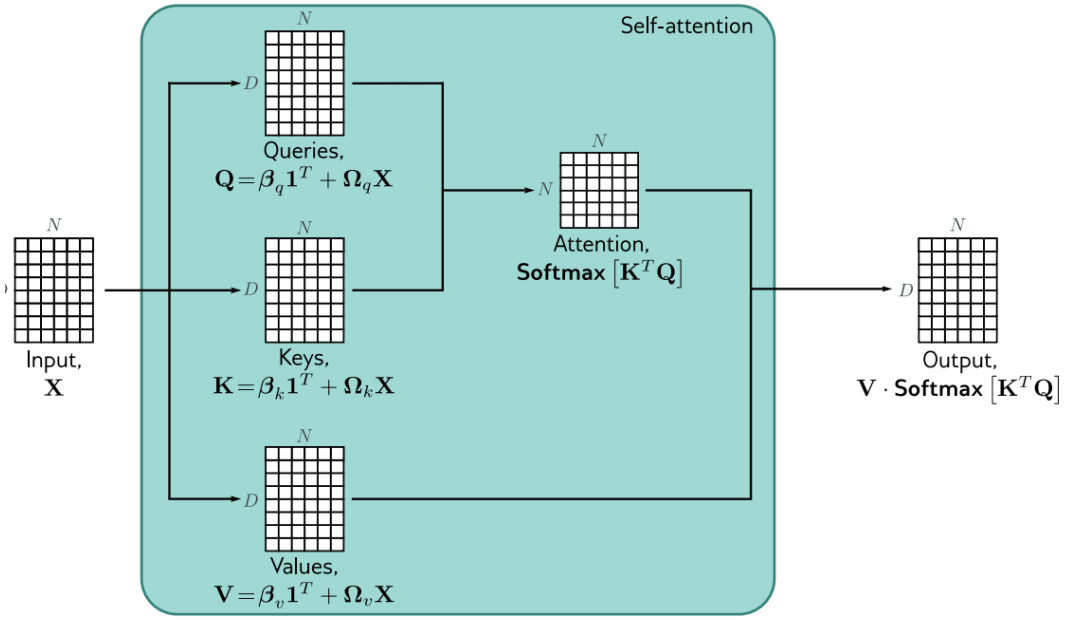

Figure 3. Transformer Model Architecture. The self-attention mechanism generates query Q, key K, and value V vectors from the input X. Attention is computed by applying Softmax to the scaled dot product of Q and K, and then weighting the results with V to capture semantic and contextual relationships between tokens. Note: this figure uses capital letters for vectors (e.g., Q, K, V), whereas vectors are denoted with arrows (e.g., $\vec{q}_i, \vec{k}_i, \vec{v}_i$) throughout the article. Adapted from: [3]

Although there is no activation function, the attention mechanism exhibits non-linear behavior due to the application of the non-linear Softmax function to compute attention weights.

The mechanism satisfies the foundational requirements of an efficient LLM, proposed in the introduction. Firstly, it uses a single shared set of parameters across all computations, defined as:

$$\Phi = \{\Omega_v, \Omega_q, \Omega_k\}$$

These parameters are independent of the number of input tokens. Hence, addressing the challenge of varying lengths and sizes of text input. Furthermore, the attention mechanism by design solves the problem of capturing long-range semantic dependencies within a text, a critical component to understand natural language.

5. Conclusion

This article explored two fundamental components of the Transformer architecture. Namely, the embedding process and the attention mechanism, both highlighted in Figure 1. These components are central to the Transformer's ability to encode semantic information and capture contextual relationships, hence enabling the LLMs to process and understand natural language. Although the additional components such as the multilayer perceptron and normalization layers were not discussed, this article aims to provide a foundational understanding of the mechanism underlying LLMs. It is hoped that this insight contributes to a conceptual understanding of how Transformers capture the richness and complexity of human language.

There are multiple directions where curious readers can further explore. The following are just a few of many key extensions that are related to topics discussed in this article:

Learnable parameters Query matrix Ω_q, key matrix Ω_k, and value matrix Ω_v are composed of elements that are real numbers, commonly referred to as parameters. Initially, the parameters might be initialized with random values but are fine-tuned during the training process. The core components of this process include cost function, which measures the model's performance, and its optimization through a mechanism called back-propagation. While the details of the training process deserve a dedicated discussion, it is important to highlight that the foundation of the Transformer architecture lies in these learnable parameters [11].

Positional encoding The order of tokens in a sequence matters. For example, the sentence "John eats a burger" differs from the sentence "A burger eats John". A notable semantic difference arises, despite both sentences containing the exact same set of words. Since matrices alone do not capture positional information, positional encoding must be incorporated. Two common techniques for this are absolute positional encoding and relative positional encoding. This separation of positional information is also what enables the same weight matrices to be applied to all tokens, as each token's position is encoded independently [3].

Scaled dot-product attention In the now-famous paper titled "Attention is all you need", a technical detail is added to the Attention equation:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Where Q and K present the full array of query and key matrices, respectively. The numerator, QK^T, generates a grid of all possible dot products between pairs of keys and queries. The division by $\sqrt{d_k}$, the square root of the dimension of the query-key space, ensures numerical stability during computation [2].

I sincerely thank Prof.Dr. Ljupčo Todorovski for his exceptional guidance and insightful advice throughout the preparation of this work. I also extend my gratitude to Grant Sanderson, whose YouTube channel 3Blue1Brown provided invaluable inspiration and visual conceptual understanding. Moreover, I am grateful to the anonymous reviewer for their helpful and insightful suggestions, which significantly improved the clarity and quality of this work.

REFERENCES

[1] Google Developers, *Text Classification Guide, Word Embeddings*, (2023), URL: `https://developers.google.com/machine-learning/guides/text-classification/step-3#figure-7` (accessed 2. 1. 2025).

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gómez, Ł. Kaiser, and I. Polosukhin, *Attention Is All You Need*, arXiv preprint arXiv:1706.03762, (2017), URL: `https://arxiv.org/pdf/1706.03762` (accessed 2. 1. 2025).

[3] S. J. D. Prince, *Understanding Deep Learning*, Cambridge, MA: MIT Press, (2024), URL: `http://udlbook.com` (accessed 2. 1. 2025).

[4] *Softmax function*, Wikipedia: The Free Encyclopedia, URL: `https://en.wikipedia.org/wiki/Softmax_function` (accessed 2. 1. 2025).

[5] *Large language model*, Wikipedia: The Free Encyclopedia, URL: `https://en.wikipedia.org/wiki/Large_language_model` (accessed 2. 1. 2025).

[6] L. A. Mullen, K. Benoit, O. Keyes, D. Selivanov, and J. Arnold, *Fast, Consistent Tokenization of Natural Language Text*, The Journal of Open Source Software, 3(23), 655, DOI: `10.21105/joss.00655`.

[7] M. Dehouck, *Challenging the "One Single Vector per Token" Assumption*, Proceedings of the 27th Conference on Computational Natural Language Learning (CoNLL), Singapore, Singapore, (2023), 498–507. URL: `https://hal.science/hal-04334826` (accessed 2. 1. 2025).

[8] R. P. Lebret, *Word Embeddings for Natural Language Processing*, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, (2016), URL: `https://infoscience.epfl.ch/server/api/core/bitstreams/2244fe89-5493-4ab5-81cb-ae466fc37cc7/content` (accessed 2. 1. 2025).

[9] D. Jurafsky and J. H. Martin, *Word Meaning: Vector Semantics & Embeddings*, Stanford University, (2024), URL: `https://web.stanford.edu/~jurafsky/slp3/slides/vectorsemantics2024.pdf` (accessed 2. 1. 2025).

[10] *Dot product*, Wikipedia: The Free Encyclopedia, URL: `https://en.wikipedia.org/wiki/Dotproduct` (accessed 2. 1. 2025).

[11] Z. Zheng, H. Liang, V. Snášel, V. Latora, P. Pardalos, G. Nicosia, and V. Ojha, *On Learnable Parameters of Optimal and Suboptimal Deep Learning Models*, In Proceedings of the 31st International Conference on Neural Information Processing (ICONIP 2024), arXiv:2408.11720, URL: `https://doi.org/10.48550/arXiv.2408.11720` (accessed 2. 1. 2025).

[12] OpenAI, *ChatGPT Language Model (GPT-4o)*, used for phrasing suggestions. Accessed via `https://chat.openai.com` between December 2024 and April 2025.