CRC Credit Research Centre

CRC Working Papers

The Simple Mathematics of Large Language Models

By Joseph L. Breeden

January 2026

Keywords: Large Language Models, Transformers, Attention Models

THE UNIVERSITY of EDINBURGH

Contact
Credit Research Centre
University of Edinburgh Business School
29 Buccleuch Place
Edinburgh, EH8 9JS
credit.research.centre@ed.ac.uk

The Simple Mathematics of Large Language Models

Joseph L. Breeden
Deep Future Analytics, LLC
breeden@deepfutureanalytics.com

9 December 2025

Abstract

This document explains what large language models are and how they work, intended for readers with backgrounds in linguistics, statistics, or mathematics. This document builds the explanation from first principles, using only standard mathematical and statistical terminology. Each technical concept, whether from linguistics, mathematics, or statistics, is defined when it first appears. The goal is genuine understanding: not just what operations these models perform, but why those operations make sense.

Contents

1 Introduction: The Problem We Are Trying to Solve

After decades studying nonlinear systems and modeling (now called data science), statistics, and linguistics, I thought that I would be able to read and understand the structure of large language models. As occurs too often, I was thwarted by jargon that had no basis in any of the fields one would assume led to their development. A token is not a "query". Linguists do not speak about "keys". What is a "head"? I could guess at the origins from computer science, but even "jargon free" explainers immediately started with "query", "key", and "value". So let's start fresh. This document is a re-explanation of large language models for those with backgrounds in linguistics, statistics, and mathematics. I still need terms for the various components, but I have attempted to choose a terminology that is functional rather than anthropomorphized and anecdotal. Of course, I provide references back to the jargon of LLMs.

A large language model is a system that, given some text, assigns probabilities to what word might come next. Given "The cat sat on the," the model might assign high probability to "mat," "floor," or "chair," and low probability to "democracy" or "purple."

This is useful because language is predictable, not perfectly, but substantially. If you read "The patient was diagnosed with a rare form of," you can anticipate that a disease name is coming. This predictability reflects the structure of language and, indirectly, the structure of the world that language describes.

The engineering achievement of modern language models is that they learn this predictability from text alone, without being told the rules of grammar or the facts of the world. They are trained by showing them enormous amounts of text (trillions of words) and adjusting their internal parameters to make correct predictions. After this training, they can generate coherent text, answer questions, and perform tasks that seem to require understanding. Of course, this data-only approach also brings limitations, sometimes severe. Humans learn language from its context within the world, not just as a sequence of sounds. Animals clearly communicate, but often without an identifiable grammar, or even sound. Improving beyond the technology of LLMs will almost certainly require learning language within this greater context, but that is the topic for another paper.

This document explains the mechanism by which LLMs accomplish this word prediction task. We will see that the core ideas are not exotic: conditional probability, weighted averages, and learned linear transformations. The sophistication lies in how these simple pieces are composed.

1.1 What This Document Assumes

This article assumes familiarity with:

- Vectors and matrices: what they are, how to multiply them
- Probability: conditional probability $P(A \mid B)$, probability distributions
- The idea that statistical models have parameters fitted to data

No prior knowledge of neural networks, machine learning, or computational linguistics is assumed. All necessary concepts will be introduced and explained.

2 Text as Data: Sequences of Discrete Symbols

2.1 Tokens: The Atoms of Text

For computational purposes, text is represented as a sequence of discrete symbols called **tokens**. A token is typically a word or a piece of a word.

Why pieces of words? Consider the word "unhappiness." A vocabulary containing every possible English word would be enormous and could not handle novel words or names. Instead, modern systems use a vocabulary of about 50,000 tokens that includes common words ("the," "cat"), word fragments ("un-," "-ness," "-ing"), and individual characters. The word "unhappiness"

might be represented as three tokens: "un" + "happi" + "ness." Because modern text can be filled with emojis, mathematical symbols, and other pictorials, tokens can extend well beyond words.

We denote the vocabulary (the set of all possible tokens) as $\mathcal{V}$, and a text as a sequence:

$$(w_1, w_2, \ldots, w_n), \quad \text{where each } w_i \in \mathcal{V}.$$

2.2 The Statistical Objective

We want to model the probability of sequences. By the chain rule of probability (a basic identity, not an assumption) any joint distribution factors as:

$$P(w_1, w_2, \ldots, w_n) = P(w_1) \cdot P(w_2 \mid w_1) \cdot P(w_3 \mid w_1, w_2) \cdots P(w_n \mid w_1, \ldots, w_{n-1}).$$

Each factor is a conditional distribution: the probability of the next word given all preceding words. A language model is a system that computes these conditional distributions.

2.3 Why This Is Hard

The challenge is that the number of possible contexts grows exponentially with length. If the vocabulary has 50,000 tokens, there are $50,000^{10}$ possible 10-word contexts—more than the number of atoms in the observable universe.

We cannot store a separate probability distribution for each possible context. Instead, we need a function that takes any context and produces a probability distribution over the next word. This function must generalize: having seen "The cat sat on the mat," it should have learned something applicable to "The dog lay on the rug."

The question is: what kind of function can do this?

Note that this extends well beyond cataloging N-grams. Databases exist from which you can look up all possible one-word completions to "The cat sat on the". Randomly choosing the next word based upon the N-gram probabilities would be equivalent to a "stochastic parrot". While this dismissive description was applied to LLMs, it is incorrect. An LLM chooses the sentence completion by scanning a much larger body of prior text. LLMs do not have a true world model with which to reason, but if the prior phrase said, "Genie explained that she could not clean the floor, because...", the completion becomes fairly unambiguous. That assumes that we have an algorithm that can connect "cleaning the floor" and "cat" to a mat.

It is worth noting that despite their sophistication, large language models remain finite-order Markov models in a precise sense: they have a hard-coded maximum context length, and everything before that cutoff is ignored (Shalizi, 2023). The neural network architecture performs implicit smoothing across contexts—learning to generalize from seen contexts to unseen ones—but the fundamental limitation remains. The model cannot, in principle, capture dependencies that span longer than its context window, no matter how much data it has seen. Of course, many methods are being explored to work around this fixed window view of the world.

LLMs are a classic example of exhibiting emergent phenomena. Nothing in the math that follows explicitly encodes such relationships, but the algorithms exhibit behaviors that can be described with higher layers of abstractions. Specifically, to create the best possible prediction of the next word, the algorithm appears to learn "concepts" and the relationships between concepts such as have been laboriously curated in databases such as WordNet. LLMs appear to be both more and less than these prior curated efforts. Any popular LLM today goes well beyond the lists of concepts and relationships found in WordNet (Miller, 1995), and yet LLMs are more error prone in these relationships than the rigorous editing of WordNet. No one has solved how to directly encode curated knowledge from a database like WordNet into an LLM. To do so might save a moderate-sized country's power consumption.

3 Representing Tokens as Vectors

3.1 The Problem with Treating Words as Unrelated Symbols

The simplest representation of a token is an arbitrary label: token 1, token 2, and so on. Mathematically, we could represent the i-th token in the vocabulary as a "one-hot" vector, a vector of length $|\mathcal{V}|$ with a 1 in position i and 0s elsewhere.

This representation treats all tokens as equally different from each other. But "cat" and "dog" share more in common (both animals, both pets, both nouns) than "cat" and "although." A useful representation should capture these similarities.

3.2 Words as Points in Space

The essential idea is to represent each token as a point in a continuous space of moderate dimension (typically 1,000 to 10,000 dimensions). Tokens with similar meanings or grammatical functions are placed near each other; dissimilar tokens are far apart.

Formally, we define a function:

$$E : \mathcal{V} \to \mathbb{R}^d$$

that assigns to each word (token) w a vector $\boldsymbol{e}_w \in \mathbb{R}^d$. The coordinates of these vectors are parameters—numbers that are learned from data.

This representation is sometimes called an **embedding** because it embeds the discrete set of words into a continuous vector space. This d-dimensional space is not a one-hot encoding, but we have also not said how d is chosen or what these dimensions are. As will be seen, the embedding functions like principal component analysis (PCA) in statistics, compressing a large set of attributes into a much smaller set of features that may, or may not, be interpretable intuitively.

3.3 Why This Helps

Once tokens are vectors, we can use the tools of linear algebra. Tokens can be added, averaged, compared using distances and angles. A model that has learned something about "cat" can apply that knowledge to "dog" if their vectors are similar.

This idea has a linguistic foundation. In 1957, the linguist J.R. Firth articulated a principle that has become foundational: "You shall know a word by the company it keeps" (Firth, 1957). Words that appear in similar contexts tend to have similar meanings. The modern approach learns token vectors by exploiting exactly this regularity: tokens that can substitute for each other in many sentences end up with similar vectors. The influential word2vec system (Mikolov et al., 2013) demonstrated that such learned vectors capture semantic relationships with remarkable fidelity. Famously, vector arithmetic like king $-$ man $+$ woman $\approx$ queen often holds.

4 The Central Problem: Meaning Depends on Context

4.1 The Same Word Can Mean Different Things

A single vector per word cannot capture the full complexity of language. Consider the word "bank":

- "I deposited money at the bank." (a financial institution)
- "We walked along the river bank." (the land beside a river)
- "The plane banked sharply left." (tilted)

This phenomenon (a single word having multiple distinct meanings) is called **polysemy**. It is pervasive in natural language. The word "run" has dozens of meanings; "set" has over 400 dictionary entries.

A fixed vector for "bank" cannot simultaneously be close to "money" and "river" and "tilt." The meaning depends on context.

4.2 The Computational Goal

We need to compute **context-dependent** representations. The word "bank" should have different vector representations in different sentences, with the representation shaped by the surrounding words.

Given a sequence of word vectors $(\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n)$, we want to compute a new sequence of vectors $(\boldsymbol{h}_1, \ldots, \boldsymbol{h}_n)$ where each $\boldsymbol{h}_i$ incorporates information from the context—the surrounding words.

How do we do this? The next section develops the mechanism.

5 The Core Mechanism: Learned Weighted Averaging

We now arrive at the heart of modern language models. The mechanism is, mathematically, straightforward: a weighted average of vectors, where the weights are themselves computed from the data. The sophistication is in how the weights are determined.

5.1 The Idea of a Weighted Average

Suppose we have vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ and we want to combine them into a single vector. The simplest approach is an unweighted average:

$$\bar{\boldsymbol{v}} = \frac{1}{n}(\boldsymbol{v}_1 + \boldsymbol{v}_2 + \cdots + \boldsymbol{v}_n).$$

A weighted average allows different vectors to contribute differently:

$$\bar{\boldsymbol{v}} = \alpha_1 \boldsymbol{v}_1 + \alpha_2 \boldsymbol{v}_2 + \cdots + \alpha_n \boldsymbol{v}_n,$$

where the weights α_j are non-negative and sum to 1.

If we are computing a context-dependent representation for position i, we want the weights to depend on i. Different positions should draw on different parts of the context. So we write:

$$\boldsymbol{h}_i = \sum_{j=1}^{n} \alpha_{ij} \boldsymbol{v}_j,$$

where α_{ij} is the weight that position i places on position j.

The key question is: how do we determine the weights α_{ij}?

Before answering this, it helps to keep the ultimate goal in mind. After many layers of such weighted averaging (and other transformations), the model will produce a final representation for the last position in the sequence. This final vector will then be compared to learned vectors for every word in the vocabulary, and the word whose vector aligns most closely will receive the highest probability. The weighted averaging mechanism is the means by which context shapes this final representation.

5.2 The Concept of Influence

The weight α_{ij} determines how much position j **influences** the new representation at position i. This is a directional relationship: the influence of j on i can be entirely different from the influence of i on j.

Consider the sentence "The cat sat on the mat." When computing the representation of "sat":

- "cat" should have high influence (it is the subject. The one doing the sitting).
- "mat" should have moderate influence (it is the location).

• "the" should have low influence (grammatical function word, not central to meaning).

When computing the representation of "cat":

• "sat" provides information about what the cat is doing.

• "mat" provides information about the scene.

The influence is asymmetric: the relationship of "cat" to "sat" differs from that of "sat" to "cat."

We will use the term **influence score** for the raw measure of how much j should influence i, and **influence weight** for the normalized version (non-negative, summing to 1) used in the weighted average. IIn the standard literature, these are called *attention scores* and *attention weights*, respectively. This article avoids anthropomorphized terms such as "attention", because they borrow metaphors that can prove a hinderance to contributions from scientists in other fields, although attention is not nearly a obstructive as Query, Key, and Value, which will be discussed later. This article is attempting an intuitive yet boringly technical description of the essential terms. The mechanism is simply weighted averaging with learned weights indicating a relationship between tokens.

5.3 Computing Influence Scores

We measure influence using the **dot product**. Given two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, their dot product is:

$$\boldsymbol{a} \cdot \boldsymbol{b} = \sum_{k=1}^{d} a_k b_k = |\boldsymbol{a}| \cdot |\boldsymbol{b}| \cdot \cos\theta,$$

where θ is the angle between them. The dot product is large when vectors point in similar directions (small angle, $\cos\theta \approx 1$) and small or negative when they point in different directions.

We need a measure of association between vectors that can be computed efficiently and that is differentiable (so we can learn the parameters by gradient-based optimization). The dot product is the simplest such measure. It is linear in each argument, which makes the mathematics tractable. Cosine similarity is an intuitively satisfying measure that has arisen independently in many contexts. I also derived this measure a decade ago for a wine name matching project, only to find that it already had a name.

5.4 Why We Need Separate Projections

We could compute influence directly as $\boldsymbol{e}_i \cdot \boldsymbol{e}_j$, the dot product of the original word vectors. But this is too limited for two reasons.

First, as mentioned earlier, influence is asymmetric. The way "cat" influences "sat" differs from how "sat" influences "cat." If we used $\boldsymbol{e}_i \cdot \boldsymbol{e}_j$, we would get symmetric influence scores (since the dot product is symmetric: $\boldsymbol{a} \cdot \boldsymbol{b} = \boldsymbol{b} \cdot \boldsymbol{a}$).

Second, what determines influence may differ from what information is transferred. The features that determine how much j should influence i may differ from the features that should actually be aggregated into the new representation.

The solution is to use three separate linear transformations. Given a vector $\boldsymbol{e} \in \mathbb{R}^d$, we compute $\boldsymbol{W}\boldsymbol{e} \in \mathbb{R}^k$, where $\boldsymbol{W}$ is a $k \times d$ matrix of learned parameters.

What does this operation do, intuitively? The original vector $\boldsymbol{e}$ is quantified according to its values in a d-dimensional feature space and encodes many things simultaneously: syntactic role, semantic content, and other features we may not have names for. This is the original embedding. Each of the k rows of $\boldsymbol{W}$ acts as a feature detector: the dot product of a row with $\boldsymbol{e}$ measures how strongly $\boldsymbol{e}$ exhibits the feature that row represents. The output $\boldsymbol{W}\boldsymbol{e}$ is a list of k such measurements—a set of learned features extracted from the original representation.

Recall that we are computing a new representation for position i by aggregating information from context positions j. There is one position i being updated, and many positions j that may

influence it. We define three transformations:

$$\boldsymbol{a}_i = \boldsymbol{W}^A \boldsymbol{e}_i \qquad \text{(applied to the single position } i \text{ being updated)} \tag{1}$$

$$\boldsymbol{b}_j = \boldsymbol{W}^B \boldsymbol{e}_j \qquad \text{(applied to each context position } j\text{)} \tag{2}$$

$$\boldsymbol{c}_j = \boldsymbol{W}^C \boldsymbol{e}_j \qquad \text{(also applied to each context position } j\text{)} \tag{3}$$

In the standard literature, these three transformations are called **Query** ($\boldsymbol{W}^A$), **Key** ($\boldsymbol{W}^B$), and **Value** ($\boldsymbol{W}^C$). These terms are borrowed from database retrieval: one queries a database by matching keys to retrieve values. The analogy is imperfect. Position i does not "query" anything; it has a vector representation that is compared to other vector representations. Nothing is being "looked up" in the sense of database retrieval. Whatever the original motivation, the terminology has become standard without offering much insight into the mechanism. At least for myself, it was clearly a barrier, because there is no database, so what is being queried? This article uses the neutral notation $\boldsymbol{W}^A$, $\boldsymbol{W}^B$, $\boldsymbol{W}^C$ and describe them by their computational roles.

$\boldsymbol{W}^B$ and $\boldsymbol{W}^C$ answer different questions about each context position j:

- $\boldsymbol{W}^B$ extracts features that determine: *How much should j influence i?* These features are used to compute the influence weight.
- $\boldsymbol{W}^C$ extracts features that determine: *If j does influence i, what information should it contribute?* These features are what actually get aggregated in the weighted sum.

These can differ. Consider "The cat sat on the mat" when computing the representation for "sat":

- The word "cat" should have high influence. It is the subject of the verb.
- What makes "cat" relevant to "sat"? Syntactic features: it is a noun in subject position.
- What information should "cat" contribute? Semantic content: that the subject is an animal, small, domestic, and so on.

The features that signal "I am relevant to this verb" need not be the same as the features that encode "I am a cat." The separation of $\boldsymbol{W}^B$ and $\boldsymbol{W}^C$ allows the model to learn this distinction.

The matrices $\boldsymbol{W}^A$, $\boldsymbol{W}^B$, and $\boldsymbol{W}^C$ are learned from data. When we say that $\boldsymbol{W}^B$ "extracts features relevant to influence," we are describing what these matrices do after training. Initially, they are filled with random numbers and extract nothing meaningful. Through exposure to trillions of tokens, repeatedly adjusting the matrix entries to improve predictions, the model discovers which linear combinations of features serve each role. The structure is found, not specified.

The first two transformations, $\boldsymbol{W}^A$ and $\boldsymbol{W}^B$, are used to compute influence weights, as we describe next. The third, $\boldsymbol{W}^C$, is used in the subsequent aggregation step. It extracts the content that will actually be combined once the weights are determined.

For readers from fields such as demography, actuarial science, or credit risk modeling, which includes myself, the logic will be familiar. Consider Age-Period-Cohort (APC) decomposition: one defines vectors of unknown parameters along three dimensions (age, vintage, and time) together with an additive structure combining them to generate forecasts. The parameters are not specified by the analyst; they are solved for by optimizing a likelihood function, typically via constrained gradient descent. The resulting nonparametric functions may or may not correspond to identifiable real-world features. In practice, interpretability is uneven.

Large language models follow the same logic at vastly greater scale. We define a mathematical structure (embedding vectors, transformation matrices, their composition), then optimize over trillions of examples to find parameter values that predict well. The resulting matrices encode something about language, but whether that something aligns with human-interpretable concepts like syntax or semantics is, as in APC, a mixed proposition.

The dimensions d and k are design choices made by the model architect. The embedding dimension d controls the expressiveness of word representations. Typical values range from 768 to 16,384. The projection dimension k is typically set to d/H, where H is the number of rela-

tions (see Section 6), keeping the total computation manageable. These choices are determined empirically, guided by experimentation and computational constraints.

The influence score is then:

$$r_{ij} = \boldsymbol{a}_i \cdot \boldsymbol{b}_j$$

This measures how well what position j offers (as characterized by $\boldsymbol{b}_j$) aligns with what position i is receptive to (as characterized by $\boldsymbol{a}_i$). Because $\boldsymbol{W}^A$ and $\boldsymbol{W}^B$ are different matrices, this measure is asymmetric: $r_{ij} \neq r_{ji}$ in general.

5.5 A Note on Bilinear Forms

Mathematicians may recognize that the influence score

$$r_{ij} = \boldsymbol{a}_i \cdot \boldsymbol{b}_j = (\boldsymbol{W}^A \boldsymbol{e}_i)^\top (\boldsymbol{W}^B \boldsymbol{e}_j) = \boldsymbol{e}_i^\top (\boldsymbol{W}^A)^\top \boldsymbol{W}^B \boldsymbol{e}_j$$

is a **bilinear form** in $\boldsymbol{e}_i$ and $\boldsymbol{e}_j$.

A bilinear form is a function $\beta(\boldsymbol{x}, \boldsymbol{y})$ that is linear in each argument when the other is held fixed. If you double $\boldsymbol{x}$, the output doubles; if you add two vectors $\boldsymbol{x}_1 + \boldsymbol{x}_2$, the output is the sum of $\beta(\boldsymbol{x}_1, \boldsymbol{y})$ and $\beta(\boldsymbol{x}_2, \boldsymbol{y})$. And likewise for $\boldsymbol{y}$.

The most general bilinear form on $\mathbb{R}^d$ can be written as $\boldsymbol{x}^\top \boldsymbol{M} \boldsymbol{y}$ for some $d \times d$ matrix $\boldsymbol{M}$. In our case, $\boldsymbol{M} = (\boldsymbol{W}^A)^\top \boldsymbol{W}^B$.

The factorization into $\boldsymbol{W}^A$ and $\boldsymbol{W}^B$ is called a **low-rank parameterization**. If $\boldsymbol{W}^A$ and $\boldsymbol{W}^B$ are $k \times d$ matrices with $k < d$, then $\boldsymbol{M}$ has rank at most k. This means we are learning a bilinear form from a restricted family—one that can be expressed using fewer parameters. This reduces the number of parameters to learn and acts as a form of regularization (preventing overfitting).

For those unfamiliar with this terminology: the key point is that we are learning a flexible but structured way to measure how much one position should influence another.

5.6 A Note on Identifiability

The matrices $\boldsymbol{W}^A$ and $\boldsymbol{W}^B$ are not uniquely identified by the model's predictions (Shalizi, 2023). For any orthogonal matrix $\boldsymbol{O}$, replacing $\boldsymbol{W}^A$ with $\boldsymbol{O}\boldsymbol{W}^A$ and $\boldsymbol{W}^B$ with $\boldsymbol{O}\boldsymbol{W}^B$ yields identical influence scores:

$$(\boldsymbol{O}\boldsymbol{W}^A \boldsymbol{e}_i)^\top (\boldsymbol{O}\boldsymbol{W}^B \boldsymbol{e}_j) = \boldsymbol{e}_i^\top (\boldsymbol{W}^A)^\top \boldsymbol{O}^\top \boldsymbol{O} \boldsymbol{W}^B \boldsymbol{e}_j = \boldsymbol{e}_i^\top (\boldsymbol{W}^A)^\top \boldsymbol{W}^B \boldsymbol{e}_j.$$

Moreover, since the embedding vectors are also learned, we can make compensating changes: replacing $\boldsymbol{e}$ with $\boldsymbol{R}\boldsymbol{e}$ and the weight matrices appropriately leaves predictions unchanged for any invertible $\boldsymbol{R}$.

This non-identifiability is problematic for interpreting what the learned matrices "mean," but it is beneficial for optimization: multiple parameter configurations yield equivalent performance, creating a smooth optimization landscape. The situation is analogous to the rotation problem in factor analysis, where extracted factors are determined only up to orthogonal transformation.

5.7 From Influence Scores to Influence Weights

The influence scores $r_{ij} = \boldsymbol{a}_i \cdot \boldsymbol{b}_j$ are real numbers that can be positive, negative, or zero. We need to convert them into proper weights: non-negative numbers that sum to 1.

We first scale by dividing by $\sqrt{k}$ (where k is the dimension of the projected vectors). This prevents the dot products from becoming very large when k is large, which would cause numerical problems.

$$r_{ij} \leftarrow \frac{r_{ij}}{\sqrt{k}}$$

Then we apply the **inverse multinomial logit**:

$$\alpha_{ij} = \frac{e^{r_{ij}}}{\sum_{m=1}^{n} e^{r_{im}}}.$$

This function takes a vector of real numbers and produces a probability distribution (non-negative values summing to 1). Larger inputs receive exponentially more weight.

Statisticians will recognize this as the inverse of the log-odds transformation, generalized to more than two categories. In multinomial logistic regression, if the log-odds of category k (relative to a baseline) is z_k, then the probability of category k is $e^{z_k} / \sum_m e^{z_m}$. Here, the "log-odds of being influenced by position j" is the influence score r_{ij}, and we convert to "probability of influence" via the same transformation. In neural network terminology, this is called "softmax"—a name that describes a computational property (it is a smooth approximation to the argmax function) rather than the statistical purpose.

The inverse multinomial logit has convenient mathematical properties: it is differentiable everywhere, monotonic (higher scores give higher weights), and assigns positive weight to all positions (though the weight may be negligibly small).

5.8 The Complete Operation

Putting it all together, to compute a context-dependent representation for position i:

1. **Transform for influence computation:** Compute $\boldsymbol{a}_i = \boldsymbol{W}^A \boldsymbol{e}_i$ for the single position i being updated, and $\boldsymbol{b}_j = \boldsymbol{W}^B \boldsymbol{e}_j$ for each context position j.
2. **Compute influence scores:** For each position j, compute $r_{ij} = \boldsymbol{a}_i \cdot \boldsymbol{b}_j / \sqrt{k}$.
3. **Convert to influence weights:** Apply the inverse multinomial logit: $\alpha_{ij} = e^{r_{ij}} / \sum_m e^{r_{im}}$.
4. **Transform for content:** Compute $\boldsymbol{c}_j = \boldsymbol{W}^C \boldsymbol{e}_j$ for each context position j.
5. **Aggregate:** Compute the weighted average $\boldsymbol{h}_i = \sum_j \alpha_{ij} \boldsymbol{c}_j$.

The output $\boldsymbol{h}_i$ is a new representation for position i that incorporates information from the entire context, with each position's contribution determined by its influence weight.

6 Multiple Relations

A single set of transformation matrices $(\boldsymbol{W}^A, \boldsymbol{W}^B, \boldsymbol{W}^C)$ learns one pattern of influence. But linguistics tells us that words relate to each other in many ways:

- **Syntactic relations**: subjects relate to verbs, adjectives to nouns.
- **Semantic relations**: words relate to others with similar or associated meanings.
- **Referential relations**: pronouns relate to their antecedents. (An **antecedent** is the noun phrase that a pronoun refers to: in "Marie said she was tired," "Marie" is the antecedent of "she.")
- **Proximity relations**: adjacent words often relate grammatically.

This linguistic insight motivated an architectural choice: rather than learning a single set of transformation matrices, learn H sets in parallel. Each set $(\boldsymbol{W}_h^A, \boldsymbol{W}_h^B, \boldsymbol{W}_h^C)$ can, in principle, capture a different type of relation. Each produces its own output $\boldsymbol{h}_i^{(h)}$. These are concatenated and projected back to the original dimension:

$$\boldsymbol{h}_i = \boldsymbol{W}^O \begin{pmatrix} \boldsymbol{h}_i^{(1)} \\ \boldsymbol{h}_i^{(2)} \\ \vdots \\ \boldsymbol{h}_i^{(H)} \end{pmatrix}.$$

It is important to distinguish what is designed from what is discovered. The decision to have H parallel sets of matrices is an architectural choice. The designer adds a dimension to create capacity for multiple relations. The motivation is linguistic: language has multiple types of relations, so perhaps distinct sets of parameters can specialize to capture them. However, the design only creates the capacity for specialization. Which relations emerge, if any, is determined by training.

No one assigns roles. There is no instruction that $h = 7$ should handle pronoun reference while $h = 12$ handles subject-verb agreement. All H sets of matrices are initialized randomly and trained by the same objective: predict the next word.

Specialization emerges from optimization. If two sets learned identical patterns, one would be redundant. The model would be wasting capacity. There is implicit pressure to differentiate. Each set contributes more to reducing prediction error by capturing something the others miss. This is analogous to factor analysis, where extracted factors differentiate not because they are assigned distinct roles, but because capturing distinct sources of variation is what explains the data efficiently.

The hypothesis proved correct empirically. Models with multiple parallel sets predict better than those with one. In practice, H is typically 12 to 96.

Post-hoc analysis of trained models reveals partial alignment with linguistic intuition (Clark et al., 2019). Researchers have identified sets that appear to track syntactic dependencies, others that follow referential chains, others that emphasize adjacency, but the alignment is imperfect. The learned relations do not map cleanly onto the categories that linguists have named. Some sets capture patterns that lack obvious linguistic labels; others blend multiple linguistic functions. The model has no concept of "syntax" or "coreference." It finds patterns that reduce prediction error, and these patterns overlap with, but do not replicate, the structure that linguists have described.

In the standard literature, each parallel set is called an "attention head." This article avoids that term. The word "head" suggests a designed component with a distinct function, like the read heads of a disk drive, each positioned to access specific data. As we have seen, the parallel sets are not designed to have distinct functions. They are structurally identical, initialized randomly, and differentiate only through training. The specialization is emergent, not engineered. Calling them "heads" invites misunderstanding and has no basis in linguistics, statistics, or mathematics. This article adopts the term **relations**, acknowledging that the architectural capacity was motivated by linguistic insight, while recognizing that the learned patterns only partially align with named linguistic categories.

Viewed mathematically, each relation serves as a basis element. The H parallel sets of projection matrices define H different ways of computing influence-weighted averages. Together, they form a basis over which contextual representations are constructed. This perspective, relations as basis functions, suggests that some could be constrained to encode known linguistic structure while others remain free to discover emergent patterns. That possibility is explored in a companion paper Breeden (2025).

7 Depth: Stacking Multiple Layers

One round of weighted averaging allows each word to incorporate information from its immediate context. But some dependencies span long distances. In the sentence:

> "The scientist who discovered the high-temperature superconductor received the Nobel Prize for _____."

the blank should probably be "physics," but determining this requires connecting "Nobel Prize" back to "superconductor" and "scientist," across a long intervening clause.

To capture such dependencies, we **stack multiple layers** of the weighted averaging operation. The output of one layer becomes the input to the next:

$$\boldsymbol{h}_i^{(0)} = \boldsymbol{e}_i \qquad \text{(initial word vectors)} \tag{4}$$

$$\boldsymbol{h}_i^{(1)} = \text{Layer}_1(\boldsymbol{h}_1^{(0)}, \ldots, \boldsymbol{h}_n^{(0)})_i \tag{5}$$

$$\boldsymbol{h}_i^{(2)} = \text{Layer}_2(\boldsymbol{h}_1^{(1)}, \ldots, \boldsymbol{h}_n^{(1)})_i \tag{6}$$

$$\vdots \tag{7}$$

$$\boldsymbol{h}_i^{(L)} = \text{Layer}_L(\boldsymbol{h}_1^{(L-1)}, \ldots, \boldsymbol{h}_n^{(L-1)})_i \tag{8}$$

Each layer has its own complete set of parameters: the three transformation matrices $\boldsymbol{W}^A$, $\boldsymbol{W}^B$, $\boldsymbol{W}^C$ for each of the H relations, the output projection $\boldsymbol{W}^O$, and the feedforward transformation matrices. Nothing is shared across layers. A model with $L = 96$ layers and $H = 96$ relations per layer has 96 independent copies of the entire mechanism, each learning its own patterns. This is the origin of the enormous parameter counts: a model with $d = 12288$ dimensional embeddings, $L = 96$ layers, and $H = 96$ relations has tens of billions of parameters. Each parameter is adjusted independently during training to minimize prediction error.

Modern large language models use $L = 32$ to 128 layers. The original transformer architecture was introduced by Vaswani et al. (2017), building on earlier work on attention mechanisms by Bahdanau et al. (2015).

7.1 Nonlinear Transformations

Between layers (or within each layer), we also apply a nonlinear transformation to each position independently. This is a function applied to each vector:

$$\boldsymbol{h}_i \leftarrow \boldsymbol{W}_2 \cdot \phi(\boldsymbol{W}_1 \boldsymbol{h}_i + \boldsymbol{b}_1) + \boldsymbol{b}_2,$$

where ϕ is a nonlinear function applied element-wise.

Why nonlinearity? Without it, stacking multiple layers would be pointless: a composition of linear functions is itself linear, so multiple layers would collapse into one. The nonlinearity allows each layer to compute genuinely new features that could not be computed by a single linear transformation.

A common choice is $\phi(x) = \max(0, x)$: simply replace negative values with zero. In neural network terminology, this is called a **ReLU** (Rectified Linear Unit). The name reflects its origin in electrical engineering, where a rectifier removes the negative portion of a signal. The function is trivial to compute and works well in practice. Other choices exist. The specific form matters less than the presence of nonlinearity.

7.2 Incremental Refinement

In practice, each transformation is computed as an additive adjustment. The output is the input plus the transformation:

$$\boldsymbol{h}_i \leftarrow \boldsymbol{h}_i + f(\boldsymbol{h}_i).$$

This means each layer makes an incremental additive adjustment to the representation, rather than a holistic computation. In the neural network literature, this is called a "skip connection" or "residual connection" (He et al., 2016), again borrowing terms in some cases from electrical engineering.

This architectural choice makes the model easier to train. Training requires computing partial derivatives of the prediction error with respect to parameters in every layer, including early ones. The algorithm for this is called **backpropagation**. The chain rule is applied systematically, propagating error backward from the output through each layer. Without additive

adjustments, these multiplied derivatives can shrink exponentially, leaving early layers with negligible gradient signal—a problem known as "vanishing gradients." The additive structure ensures that derivatives always include a term of 1 from the identity path, providing a direct route for gradient flow that does not attenuate with depth.

7.3 Normalization

After each sub-operation, we typically normalize each vector to have mean 0 and variance 1 across its coordinates. This is a form of standardization that keeps the numbers in a well-behaved range during computation, preventing them from growing or shrinking uncontrollably across many layers.

7.4 The Transformer Architecture

The complete system we have described (token embeddings, position encoding, stacked layers of contextual aggregation and feedforward transformation with additive adjustments, and a final projection to vocabulary probabilities) is called a **transformer**. The name was introduced by Vaswani et al. (2017). Variants exist, including encoder-decoder architectures used for tasks like machine translation, but the dominant architecture for modern large language models is the simpler decoder-only transformer—a single stack of layers that processes and generates text one token at a time. This is the architecture we have described.

8 Position Information

There is a subtlety the preceding discussion glossed over. The weighted averaging mechanism, as described, is **symmetric with respect to position**: it treats the vectors $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n$ as a set, not a sequence. If we permuted the words, the influence scores would be the same (just reordered).

Clearly, word order is crucial. "Dog bites man" and "man bites dog" have very different meanings.

The solution is to incorporate position information into the computation. Early approaches added a position vector directly to each word embedding:

$$\boldsymbol{e}_i \leftarrow \boldsymbol{e}_i + \boldsymbol{p}_i,$$

where $\boldsymbol{p}_i \in \mathbb{R}^d$ encodes absolute position i. The vectors $\boldsymbol{p}_i$ could be learned, or defined using mathematical functions such as sines and cosines at different frequencies (Vaswani et al., 2017).

These absolute position encodings have a limitation. They struggle to generalize to sequences longer than those seen during training. Position 10,000 has a learned or computed vector, but if training never included sequences that long, the model has no experience with it.

Linguistic intuition suggests why. What usually matters is not absolute position but relative position—how far apart two words are. The relationship between a verb and its subject depends on their distance, not on whether they appear at positions 5 and 3 versus positions 500 and 498.

Current approaches encode relative position directly. The most widely adopted is **Rotary Position Embedding** (RoPE) (Su et al., 2024). Rather than adding position vectors to the embeddings, it rotates the vectors $\boldsymbol{a}_i$ and $\boldsymbol{b}_j$ by angles proportional to their positions. When the dot product $\boldsymbol{a}_i \cdot \boldsymbol{b}_j$ is computed, the rotation angles combine in a way that makes the result depend on the difference $i - j$, not on i and j individually. This approach generalizes better to longer sequences and has become standard in most recent large language models.

An alternative, **ALiBi** (Attention with Linear Biases) (Press et al., 2022), takes a simpler approach. It adds a penalty to influence scores that increases linearly with distance between positions. This requires no modification to the embeddings themselves.

The details of these schemes matter for implementation and for handling very long sequences, but the conceptual point is the same. Position information must be explicitly provided to the model, and modern approaches encode relative position rather than absolute position.

9 From Representations to Probabilities

After L layers of processing, we have a final representation $\boldsymbol{h}_n^{(L)}$ for the last position. This is converted to a probability distribution over the next token.

We compute a score for each token v in the vocabulary:

$$z_v = \boldsymbol{u}_v \cdot \boldsymbol{h}_n^{(L)},$$

where $\boldsymbol{u}_v \in \mathbb{R}^d$ is a learned vector associated with vocabulary item v. These vectors are parameters, learned during training alongside everything else.

Then we convert scores to probabilities using the inverse multinomial logit—the same operation used earlier to convert influence scores to influence weights:

$$P(w_{n+1} = v \mid w_1, \ldots, w_n) = \frac{e^{z_v}}{\sum_{v' \in \mathcal{V}} e^{z_{v'}}}.$$

This is multinomial logistic regression with $\boldsymbol{h}_n^{(L)}$ as the feature vector and $|\mathcal{V}|$ classes. The final representation encodes everything the model has extracted from the context. Tokens whose vectors $\boldsymbol{u}_v$ align with this representation receive high probability.

10 Training: Learning Parameters from Data

10.1 The Parameters

A large language model has billions of parameters. These include:

- The token embedding vectors $\boldsymbol{e}_w$ for each token $w \in \mathcal{V}$.
- The position vectors $\boldsymbol{p}_i$.
- The projection matrices $\boldsymbol{W}_{\ell,h}^A$, $\boldsymbol{W}_{\ell,h}^B$, $\boldsymbol{W}_{\ell,h}^C$ at each layer ℓ and relation h.
- The output projection matrices $\boldsymbol{W}_\ell^O$ at each layer.
- The matrices and bias vectors in the nonlinear transformations.
- The output token vectors $\boldsymbol{u}_v$ for each token $v \in \mathcal{V}$.

All of these are learned from data.

10.2 The Objective: Maximum Likelihood

Training follows the standard statistical principle of maximum likelihood optimization. Given a corpus of text, we adjust the parameters to maximize the probability the model assigns to the observed sequences.

If the training data consists of sequences, the objective is to maximize:

$$\mathcal{L}(\theta) = \sum_{\text{sequences}} \sum_{i=1}^{n} \log P_\theta(w_i \mid w_1, \ldots, w_{i-1}),$$

where θ denotes all the parameters. Taking logarithms converts products to sums and is standard in maximum likelihood.

Equivalently, we minimize the negative log-likelihood, sometimes called average surprisal, measuring how "surprised" the model is by the actual tokens. This quantity is called **cross-entropy** and is a standard measure of how well a probability distribution predicts data.

10.3 A Note on Cross-Entropy

The term **cross-entropy** comes from information theory and is worth reviewing.

Entropy, in the information-theoretic sense introduced by Claude Shannon in 1948, measures the average surprise or uncertainty in a probability distribution p:

$$H(p) = -\sum_i p_i \log p_i$$

If p is concentrated on one outcome, entropy is low—you know what is coming. If p is spread across many outcomes, entropy is high—anything could happen.

Cross-entropy measures the average surprise when outcomes are drawn from distribution p but you are using distribution q to anticipate them:

$$H(p, q) = -\sum_i p_i \log q_i$$

These are related by:

$$H(p, q) = H(p) + D_{KL}(p\|q)$$

where $D_{KL}(p\|q)$ is the Kullback-Leibler divergence—a measure of how different q is from p. Cross-entropy equals entropy plus a penalty for using the wrong distribution. If $q = p$, cross-entropy equals entropy; if $q \neq p$, cross-entropy is larger.

In language modeling, the "true distribution" p is degenerate: it puts all its mass on the word that actually occurred (a one-hot vector). The entropy of a one-hot distribution is zero—there is no uncertainty about what actually happened. Cross-entropy therefore reduces to:

$$H(p, q) = -\log q(\text{actual word})$$

This is simply the negative log-likelihood of the observed word under the model's predicted distribution q. The information-theoretic framing adds intuition—you are measuring how surprised your model is by reality—but the mathematics is identical to maximum likelihood. Minimizing cross-entropy and maximizing likelihood are the same objective.

10.4 Optimization: Gradient Descent

The parameters are optimized using gradient descent. The idea is simple. Compute the derivative of the objective with respect to each parameter (the gradient), then adjust the parameters in the direction that improves the objective.

$$\theta \leftarrow \theta + \eta \cdot \nabla_\theta \mathcal{L}(\theta),$$

where η is a step size (learning rate).

In practice, we use stochastic gradient descent. Rather than computing the gradient over the entire corpus (prohibitively expensive), we estimate it from a random subset (a "batch") of sequences. This estimate is noisy but unbiased, and the procedure converges.

Why is gradient computation feasible? The entire model, from token vectors through all layers to output probabilities, is a composition of differentiable functions. The chain rule of calculus allows us to compute derivatives of the objective with respect to any parameter, even through dozens of layers. This is done automatically by software libraries.

If anything in LLMs seems like magic, it is that billions of parameters can be estimated simultaneously. The appearance of a single massive optimization is misleading. Training frontier LLMs involves carefully designed schedules with multiple stages. Learning rates are warmed up and decayed according to specific protocols. Training data is often ordered from simpler to more complex examples—a technique called **curriculum learning** (Bengio et al., 2009).

Most importantly, the training proceeds in distinct phases: initial pre-training on raw text to learn language patterns, followed by supervised fine-tuning on curated examples to improve usefulness, often followed by reinforcement learning from human feedback (RLHF) to align the model's outputs with human preferences (Ouyang et al., 2022). The analogy to human education is apt. Children do not learn from a single massive exposure to all knowledge, but through years of staged instruction, with foundational skills preceding advanced ones. LLM training has converged on a similar insight.

10.5 Scale of Training

Modern large language models are trained on corpora containing hundreds of billions to trillions of tokens: books, articles, websites, code, and other text. Training requires thousands of specialized processors running for weeks or months, at a cost of millions of dollars.

The result is a model that has, in a sense, read more text than any human could in a lifetime, and arguably more than the cumulative reading of all humans in history, given how much writing is created but never read again, and has adjusted its billions of parameters to predict it well.

11 What Does the Model Learn?

11.1 Emergent Structure

Analysis of trained models reveals that they learn interpretable structure, though it was never explicitly programmed.

The relations often specialize. Some learn to capture influence between syntactically related words (subjects and verbs, nouns and adjectives). Others learn to track reference, connecting pronouns to the nouns they refer to. (Recall that when "she" refers back to "Marie," this relationship is called **coreference**, and "Marie" is the **antecedent** of "she.") Still others capture positional patterns or semantic associations.

Different layers tend to encode different levels of linguistic structure. Early layers (close to the input) often represent local syntax and word categories. Later layers represent meaning, discourse structure, and relationships that span longer distances.

11.2 Knowledge and Capabilities

Trained on enough data, these models exhibit surprising capabilities. Research on scaling laws demonstrated that model performance improves predictably with increased parameters, data, and computation (Kaplan et al., 2020). The GPT-3 model (Brown et al., 2020) demonstrated that scaling to 175 billion parameters and hundreds of billions of training tokens produced capabilities that were qualitatively different from smaller models:

- They can generate coherent, contextually appropriate text.
- They can answer questions, including questions that appear to require reasoning.
- They encode factual knowledge (names, dates, relationships).
- They can translate between languages, summarize text, write code.

The apparent reasoning capability deserves scrutiny. LLMs can reproduce reasoning patterns that are well-represented in their training data. If a type of logical inference, mathematical derivation, or causal argument appears frequently in the corpus, the model learns to generate similar patterns. This is impressive and useful. However, it is not abstract reasoning. The model has no internal system for manipulating symbols according to logical rules. As a result, LLMs are brittle in ways that humans are not. Change the variable names in a logic puzzle, embed a familiar problem in an unfamiliar narrative frame, or introduce irrelevant details, and performance can collapse. A human who understands the underlying logic is not fooled by such superficial changes. An LLM, matching patterns rather than reasoning abstractly, often is. This

distinction matters. LLMs are powerful tools for tasks where relevant patterns exist in the training data, but they are not general reasoning engines.

The capabilities listed above were not explicitly programmed. They emerge from the objective of predicting the next word, given sufficient data and model capacity (Wei et al., 2022). Understanding why this happens, why prediction leads to apparent comprehension, remains an open scientific question, but not without precedent. Emergent phenomena appear across many fields. Thermodynamics emerges from statistical mechanics, fluid dynamics from molecular interactions, consciousness (presumably) from neurons, market behavior from individual transactions, and the complex behavior of flocking birds from simple local rules. In each case, macro-level phenomena arise from micro-level mechanisms in ways that are not obvious from inspecting the underlying rules. LLMs appear to be another example. The simple objective of next-token prediction, scaled sufficiently, gives rise to capabilities that were not explicitly specified.

11.3 Limitations

Despite their capabilities, these models have fundamental limitations:

- **No grounding:** The models learn from text alone. They have no sensory experience of the world, no ability to verify claims against reality.
- **Hallucination:** The models generate plausible-sounding text even when it is false. They are optimized for likelihood, not truth. In fact, considering the vast amount of fiction and outright nonsense upon which they are trained, it is not clear that they could have a concept of truth.
- **No formal reasoning:** The models are powerful pattern matchers, but they do not perform rigorous deduction. They can assist theorem proving by suggesting tactics or conjectures, but verification requires external formal systems—another instance of LLMs serving as orchestrators rather than reasoners.
- **Context limitations:** The models can only consider a fixed window of text, set at design time—typically thousands to tens of thousands of tokens. Tokens beyond this window are simply invisible to the model.

A natural response to these limitations is to use LLMs not as complete solutions but as orchestrators: translating user intent into well-framed problems, delegating to specialized solvers (theorem provers, symbolic math systems, databases), and translating results back. Such hybrid architectures are emerging but remain less developed than the monolithic approach.

12 Mathematical Summary

For reference, this section collects the key operations in compact notation.

Input: A sequence $(w_1, \ldots, w_n)$ with each $w_i \in \mathcal{V}$.

Embedding: $\boldsymbol{h}_i^{(0)} = E(w_i) + \boldsymbol{p}_i$, where $E : \mathcal{V} \to \mathbb{R}^d$ and $\boldsymbol{p}_i \in \mathbb{R}^d$.

Contextual aggregation (at layer ℓ, relation h):

$$\boldsymbol{a}_i = \boldsymbol{W}_{\ell,h}^A \boldsymbol{h}_i^{(\ell-1)} \qquad \text{(what position } i \text{ is receptive to)} \tag{9}$$

$$\boldsymbol{b}_j = \boldsymbol{W}_{\ell,h}^B \boldsymbol{h}_j^{(\ell-1)} \qquad \text{(what position } j \text{ offers)} \tag{10}$$

$$\boldsymbol{c}_j = \boldsymbol{W}_{\ell,h}^C \boldsymbol{h}_j^{(\ell-1)} \qquad \text{(content to aggregate from } j\text{)} \tag{11}$$

$$\alpha_{ij} = \frac{\exp(\boldsymbol{a}_i \cdot \boldsymbol{b}_j / \sqrt{k})}{\sum_m \exp(\boldsymbol{a}_i \cdot \boldsymbol{b}_m / \sqrt{k})} \qquad \text{(influence weight)} \tag{12}$$

$$\boldsymbol{o}_i^{(h)} = \sum_{j \le i} \alpha_{ij} \boldsymbol{c}_j \qquad \text{(weighted average)} \tag{13}$$

Combining relations:

$$\tilde{\boldsymbol{h}}_i = \boldsymbol{W}_\ell^O[\boldsymbol{o}_i^{(1)}; \ldots; \boldsymbol{o}_i^{(H)}]$$

Residual connection and nonlinearity:

$$\boldsymbol{h}_i^{(\ell)} = \text{Normalize}\big(\boldsymbol{h}_i^{(\ell-1)} + \tilde{\boldsymbol{h}}_i + \boldsymbol{W}_2 \cdot \max(0, \boldsymbol{W}_1(\boldsymbol{h}_i^{(\ell-1)} + \tilde{\boldsymbol{h}}_i))\big)$$

Output:

$$P(w_{n+1} = v \mid w_1, \ldots, w_n) = \frac{\exp(\boldsymbol{u}_v \cdot \boldsymbol{h}_n^{(L)})}{\sum_{v'} \exp(\boldsymbol{u}_{v'} \cdot \boldsymbol{h}_n^{(L)})}$$

Training:

$$\max_\theta \sum_{\text{sequences}} \sum_{i=1}^{n} \log P_\theta(w_i \mid w_1, \ldots, w_{i-1})$$

13 Conclusion

A large language model is a conditional probability distribution over words, given context. The core mechanism is learned weighted averaging, computing context-dependent representations by averaging over the context, with weights determined by influence scores between positions.

The influence is measured as a dot product after learned linear projections—a bilinear form with learned parameters. This allows the model to learn, from data, which positions should influence which, and how strongly. Stacking many layers of this operation, with multiple parallel operations per layer, allows the model to learn complex relationships at multiple levels of abstraction.

The mathematics is not exotic: linear algebra (matrix multiplication, dot products), probability (conditional distributions, the normalized exponential), and optimization (gradient descent). What is remarkable is that these simple components, composed and scaled, produce systems that predict language well enough to exhibit capabilities that appear to form concepts and identify complex relationships.

The hope is that this exposition has made the mechanism clear—not as a black box, but as a comprehensible, if complex, mathematical object.

Before writing this document, I found the standard literature on large language models nearly impenetrable, despite decades of experience with nonlinear systems, statistics, and linguistics. The barrier was not the mathematics, which turns out to be straightforward, but the terminology. The field has accumulated an amorphous collection of jargon borrowed from disparate domains (databases, cognitive science, electrical engineering) largely disconnected from the underlying mathematics and statistics. A token is not a "query." Linguists do not speak of "keys." The term "head" suggests engineered specialization where none exists. This terminology obscures rather than illuminates. My hope is that this document demonstrates that the core ideas can be explained in the standard language of mathematics and statistics, and that doing so makes the mechanism not just accessible but genuinely comprehensible to the uninitiated.

References

Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*.

Bengio, Y., Louradour, J., Collobert, R., & Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*, pages 41–48.

Breeden, J. L. (2025). Concept-aligned language models: Grounding representations in semantic ontologies. *Preprint*.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33, 1877–1901.

Clark, K., Khandelwal, U., Levy, O., & Manning, C. D. (2019). What does BERT look at? An analysis of BERT's attention. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 276–286.

Firth, J. R. (1957). A synopsis of linguistic theory, 1930–1955. In *Studies in Linguistic Analysis*, pages 10–32. Blackwell, Oxford.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., & Amodei, D. (2020). Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems*, 26, 3111–3119.

Miller, G. A. (1995). WordNet: A lexical database for English. *Communications of the ACM*, 38(11), 39–41.

Shalizi, C. R. (2023). "Attention", "Transformers", in neural network "Large Language Models". Online notebook, `http://bactra.org/notebooks/nn-attention-and-transformers.html`. First version March 2023, last updated January 2025.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. (2022). Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35, 27730–27744.

Press, O., Smith, N. A., & Lewis, M. (2022). Train short, test long: Attention with linear biases enables input length extrapolation. In *Proceedings of the 10th International Conference on Learning Representations (ICLR)*.

Su, J., Lu, Y., Pan, S., Murtadha, A., Wen, B., & Liu, Y. (2024). RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568, 127063.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 5998–6008.

Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., et al. (2022). Emergent abilities of large language models. *Transactions on Machine Learning Research*.