From Text To Attention

How Transformers See Language

Part 1

Connecting conceptual foundations to mechanical implementation.

Bridging the Knowledge Gap

Filling the gaps in our understanding of Neural Networks

What we know

- ✓ Neural networks process numbers
- ✓ Models learn by adjusting weights
- ✓ Attention focuses on relevant words

The missing pieces

- ? How does text transform into numbers?
- ? What does the model actually "see"?
- ? How does attention work mechanically?

The Fundamental Challenge

Neural networks only understand **NUMBERS**.

Human language is **TEXT**.

We need a translation layer that converts text into numbers while preserving semantic relationships.

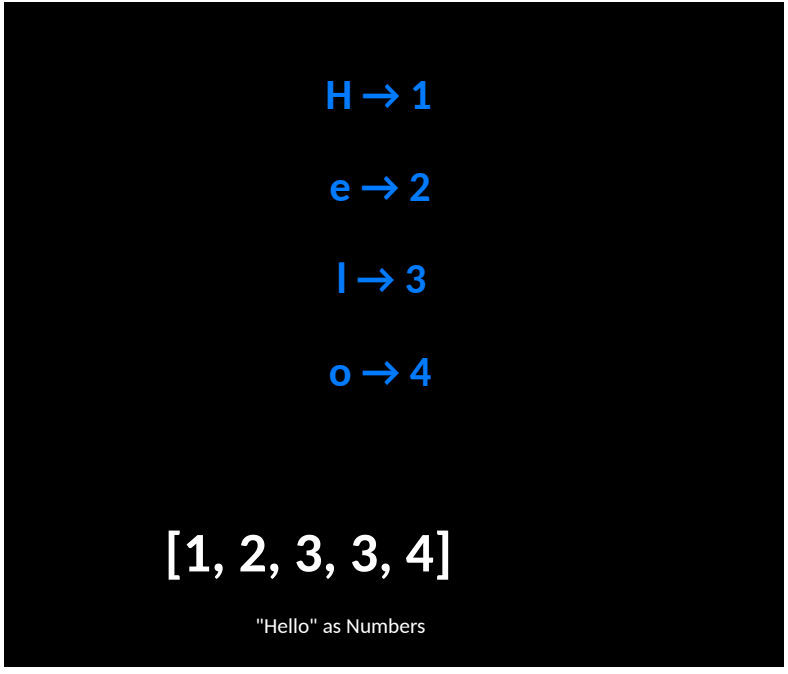
So… how?

The Character-Level Approach

Idea 1: One number per character

The most basic way to achieve numerical representation is to assign a **unique integer** to every individual character in the alphabet.

While simple, this approach operates at the smallest possible unit of language, ignoring words and concepts entirely.



Why Character-Level Models Struggle

The hidden cost of extreme granularity

The Step Problem

A word like **"understanding"** needs **13 separate steps** to process — the model must work many times harder to see one concept.

Manual Pattern Learning

The model must learn prefixes and roots from individual characters rather than receiving those units directly.

High Cognitive Load

It's like teaching reading by showing letters only — building grammar and meaning becomes much harder.

Extreme Inefficiency

Large amounts of data are spent learning basic vocabulary instead of higher-level logic.

The Word-Level Approach

Idea 2: Assigning one unique number to every whole word

The Concept

```
"The cat sat"
↓
[1, 2, 3]
```

/ **Concept Mapping:** Each word represents a single, discrete unit of meaning.

/ Much more intuitive than character-level processing.

The Challenges

! **The "Form" Problem:** Are "cat", "cats", and "cat's" three different concepts?

! **The Unknown:** How to handle typos like "teh" or rare technical terms?

! **Out of Vocabulary:** Any word not in the training set becomes a "hole" in understanding.

The Vocabulary Explosion

Scaling challenges of word-level models

170,000+

Common English Words

3 Billion

Parameters for Lookup

Computational Bottleneck

Parameter Bloat

Each unique word needs its own embedding row; huge vocabularies create large memory overhead.

The "Unknown" Problem

Fixed vocabularies fail on names, slang, and new terms, reducing robustness.

Multilingual Scaling

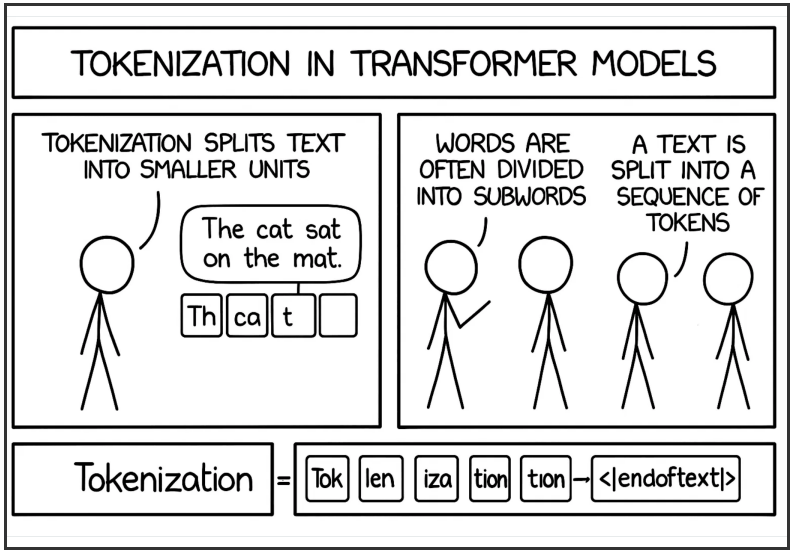
Adding languages multiplies vocabulary size, driving up compute and storage costs.

The Subword Solution

Finding the "Goldilocks" balance between characters and words

Strategic Splitting

- **The Middle Ground:** Bigger than characters (too granular) but smaller than whole words (too many).
- **Efficiency:** Common words stay whole; rare words break into recognizable, meaningful pieces.
- **Example:** "understanding" → ["under", "stand", "ing"]
- A vocabulary of 30k–100k tokens can represent almost any text in any language.



Byte Pair Encoding (BPE)

Algorithmic construction of modern tokenizers

1. Initialization

Begin with each character as an individual token.

2. Iterative Merging

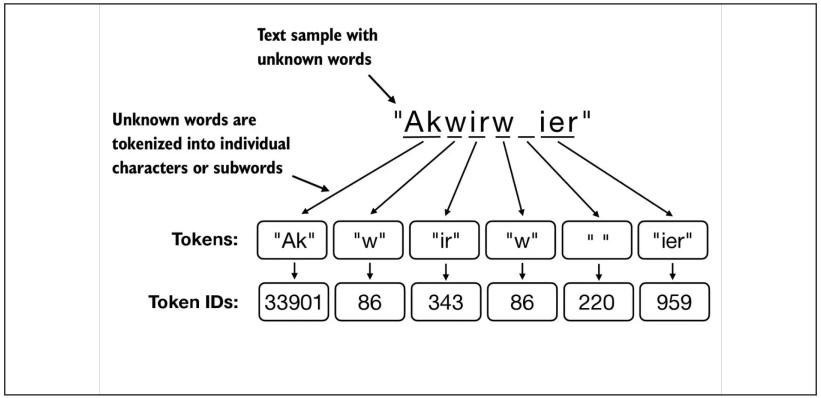
Merge the most frequent adjacent token pair into one.

3. Optimization

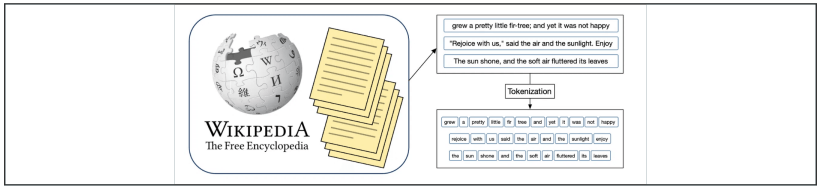
Repeat merges until reaching the target vocabulary size.

4. Self-Organizing

Frequent patterns become single tokens; rare ones stay fragmented.



Visualizing the Token Merging Process



Tokenization in Modern Practice

How GPT-style tokenizers process real-world text

Space Handling

```
"Hello world"
↓
["Hello", " world"]
```

Spaces are often attached to the beginning of the following word, not treated as separate tokens.

Contractions

```
"don't"
↓
["don", "'t"]
```

Common contractions are split into specific markers that the model has learned to associate with negation or possession.

Complex Words

```
"artificial intelligence"
↓
["art", "ificial", " intelligence"]
```

Long or rare words are broken into subword units that might not always align with linguistic syllables.

Impact: Understanding these splits is crucial for managing context limits and API costs.

Why Tokenization Matters

The practical implications of how models "see" text

Operational Impact

Context Limits

GPT-4's **128K limit** refers to tokens, not words. In English, this is roughly 1.3 tokens per word, but code can be much higher.

Financial Cost

API billing is calculated based on **token counts**, not character counts. Efficient prompting requires understanding token density.

Model Behavior

Logical Failures

Strange failures in **counting letters** or reversing words often stem from the model seeing subword units rather than individual characters.

Language Equity

Different languages require different numbers of tokens to express the same concept, affecting both **cost and performance** globally.

Summary: The Model's "Eyes"

Tokenization as the foundation of transformer processing

The Granularity Balance

Characters → Too Granular

Words → Too Many

Subwords → Just Right

The tokenizer determines the fundamental units the model uses for "thought."

Operational Impact

Poor tokenization makes the model work harder to understand context and can lead to strange failures in basic tasks.

Perception

If the model can't "see" a pattern in the tokens, it can't learn the underlying meaning of the text.

Next: Moving from arbitrary IDs to semantic meaning

From Arbitrary IDs to Meaning

Moving beyond simple integer labels

The Problem

```
"Hello world"
↓
[15496, 995]
```

These IDs are **arbitrary**. The number 15496 doesn't "know" it represents a greeting, and 995 doesn't "mean" anything about the Earth.

The Goal

We need representations that capture MEANING.

We want to transform these discrete, arbitrary integers into continuous, dense vectors where the numbers themselves encode semantic relationships.

In this new space, "Hello" and "Hi" should be mathematically similar, while "Hello" and "Banana" should be distant.

The Concept of Embeddings

Representing tokens as points in high-dimensional space

Spatial Logic

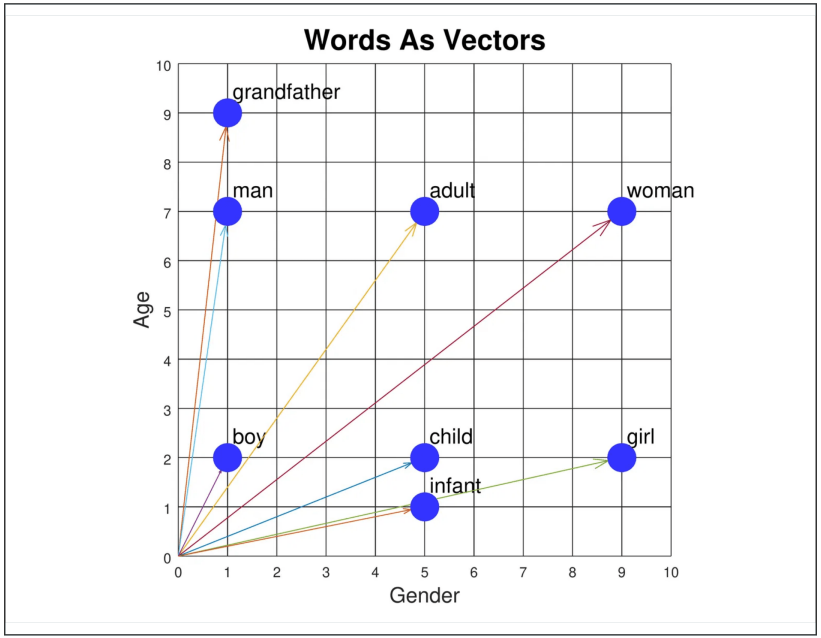
Tokens with similar meanings are placed **near each other** in the vector space. Proximity equals semantic similarity.

Vector Representation

Each token becomes a list of numbers (a vector) instead of a single ID, allowing for mathematical operations on meaning.

Multidimensionality

Modern models use **hundreds or thousands** of dimensions to capture the subtle nuances of human language.



Visualizing Semantic Space

How models organize concepts across dimensions

Clustering

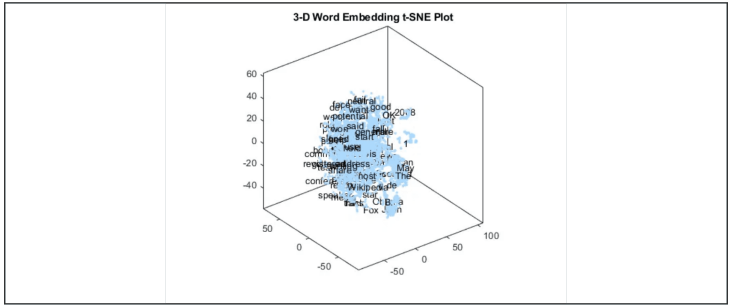
Related concepts (e.g., royalty, gender, animals) naturally group together in the high-dimensional vector space.

Emergent Properties

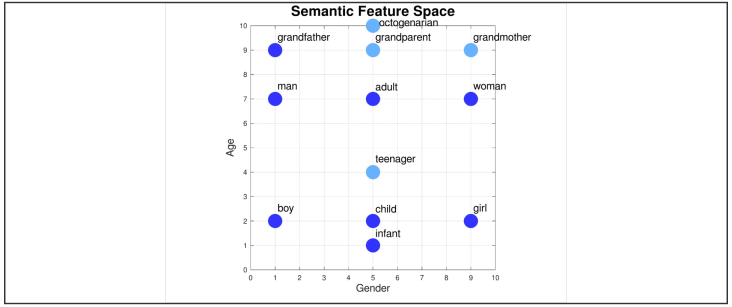
Dimensions like "gender" or "royalty" emerge automatically during training without manual labeling.

Semantic Navigation

The mathematical "distance" between points represents the semantic similarity of the tokens.



3D t-SNE Visualization of Word Clusters



2D Semantic Feature Space Mapping

Semantic Vector Arithmetic

Mathematical logic embedded in language

king - man + woman ≈ queen

Directional Meaning

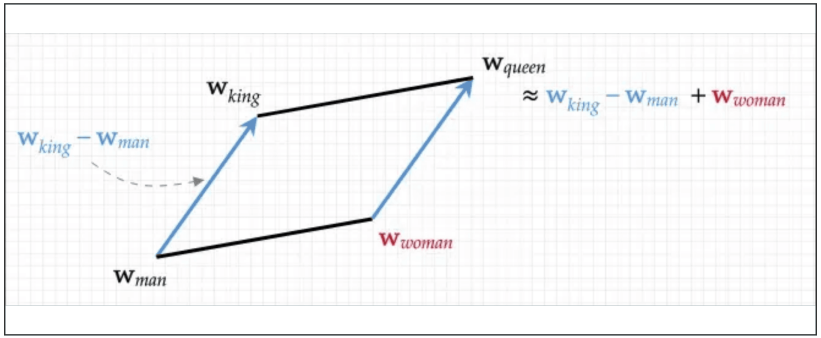
The man→woman vector mirrors king→queen.

Relational Encoding

Embeddings encode relations as consistent offsets.

Universal Patterns

These offsets recur across large vocabularies.



Visualizing Vector Logic in Semantic Space

Embeddings in the Pipeline

How tokens transform into dense vectors

The Lookup Table

```
ID 15496
↓
[0.12, -0.45, 0.88, ...]
```

Each token ID acts as an **index** to a specific row in the embedding matrix. This is a simple, fast lookup operation.

The result is a **768-dimensional vector** (for base models) that represents the token's core semantic profile.

Information Density

A single vector packs multiple layers of semantic information, capturing nuance that a single integer ID never could.

The "Raw" Starting Point

This vector is the **isolated representation** of the word. It contains the general meaning of the token before any context from the surrounding sentence is applied.

Summary: Embeddings

Converting discrete tokens into meaningful continuous space

Core Functions

- Transform arbitrary integer IDs into points in a high-dimensional semantic map.
- Ensure proximity in vector space equals similarity in meaning.
- Pack multiple layers of semantic nuance into a single dense vector.

Foundational Layer

Every transformer starts by looking up these pre-trained meanings. It is the bedrock of all subsequent processing.

The Limitation

Embeddings are "bag-of-words" by default. They represent isolated meaning but ignore the order of words.

Next: Solving the problem of sequence and word order

The Problem of Sequence

Attention is "bag-of-words" by default

Order Neutrality

"The dog bit the man"
vs.
"The man bit the dog"

Without extra information, attention treats these two sentences **identically**. The mathematical operations don't care about position.

This is known as **permutation invariance**—the model sees a set of words, not a sequence.

The Loss of Logic

In human language, **position** is often the primary driver of meaning. Order determines who did what to whom.

The Requirement

We must find a way to inject "where" a word is into "what" a word is. We need to make the model aware of the **dimension of time** in language.

Positional Encoding

Adding a sense of time and order to the model

Injecting Sequence

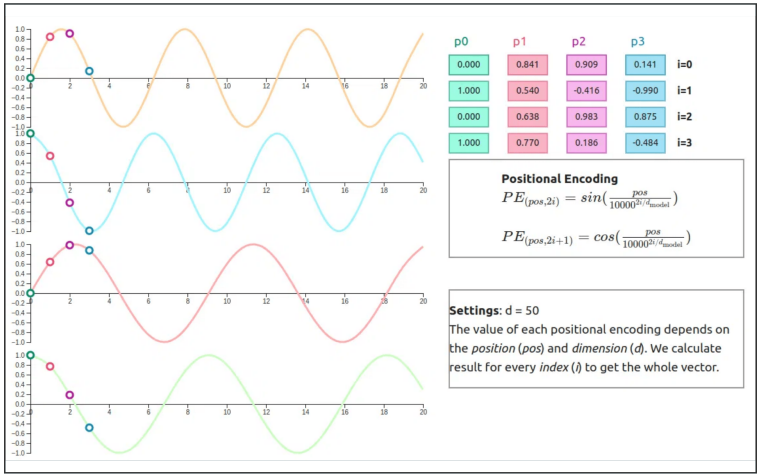
Transformers process tokens in parallel, so we add tags to show **where** each token sits in the sequence.

Vector Addition

A small positional vector is added to each embedding, encoding both **meaning** and **position**.

Spatial Awareness

This helps the model tell apart different word orders (e.g. who did what) using position tags.



How Positional Patterns Work

Using mathematical waves to encode location

Periodic Functions

We use **sine and cosine** waves of varying frequencies. Each dimension of the embedding follows a different wave.

Unique Fingerprint

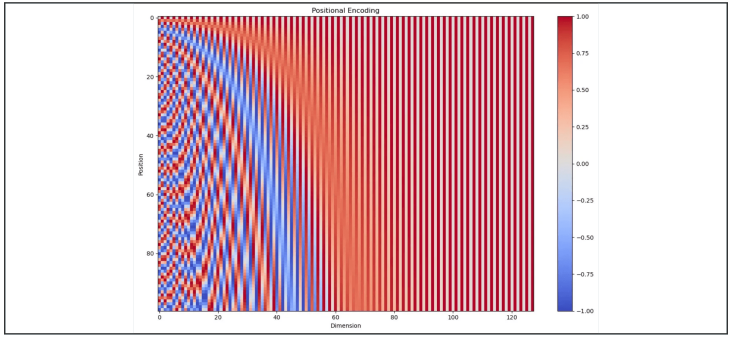
Every position in the sequence (0, 1, 2...) gets a unique combination of values, creating a "fingerprint" for that location.

Relative Distance

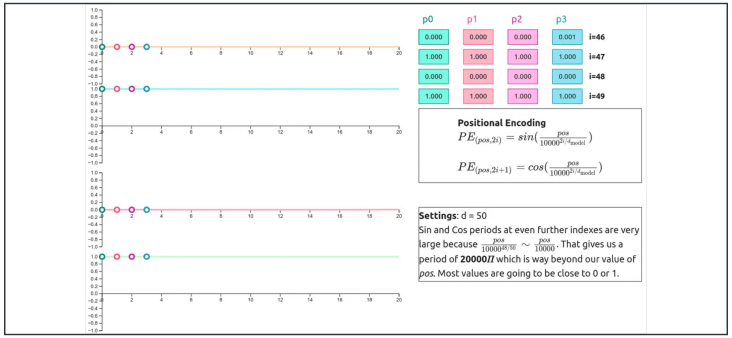
The model can calculate the **relative distance** between words because the waves are mathematically related.

Generalization

This method allows the model to handle sequences longer than those it saw during training.



Sine and Cosine Wave Patterns



Unique Encoding Matrix for Positions

Why Sine Waves?

The logic behind the original transformer's choice

Mathematical Advantages

Relative Position

For any fixed offset **k**, the position **p+k** can be represented as a linear function of position **p**. This allows the model to learn to attend by relative position easily.

Generalization

Since the waves are continuous, the model can extrapolate and handle **longer sequences** than those it saw during training.

Efficiency & Logic

No Learned Parameters

Unlike other methods, sinusoidal encoding doesn't require extra parameters to learn. The patterns are **fixed and pre-calculated**.

Unique Multi-Scale Patterns

By using multiple frequencies, the model creates a **unique fingerprint** for every single position in a very high-dimensional space.

The Pipeline So Far

From raw text to context-ready vectors

Step 1

Tokenization

Breaking raw text into discrete subword units (IDs).

Step 2

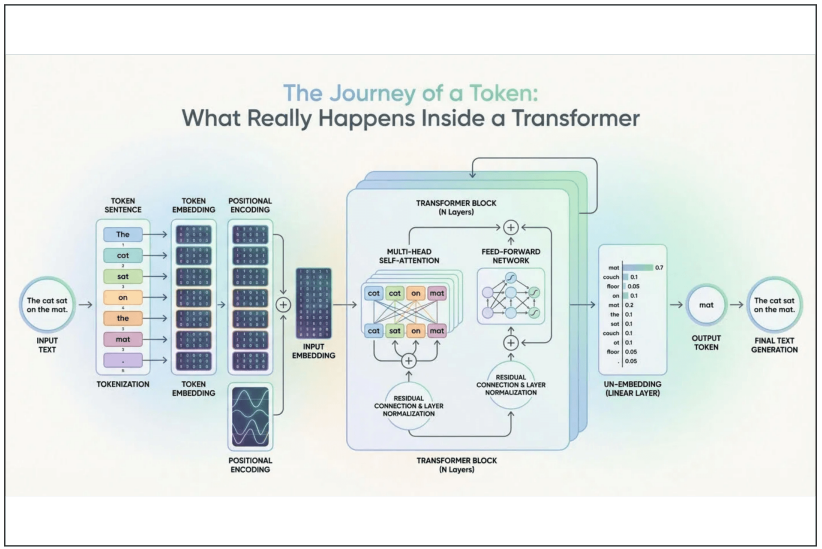
Embedding

Converting arbitrary IDs into dense semantic vectors.

Step 3

Positional Encoding

Injecting sequence awareness via mathematical wave patterns.



Summary: Positional Encoding

Injecting the dimension of time into language

Core Functions

- Inject the critical dimension of time and word order into static semantic embeddings.
- Use mathematical wave patterns to create a unique fingerprint for every position.
- Enable the model to distinguish between different word orders and logical structures.

Order-Dependent Meaning

Preserves the nuances of language where position determines roles (e.g., subject vs. object).

The Next Step

Now that each token knows "WHAT" it is and "WHERE" it is, we can move to the core of the transformer.

Next: Self-Attention—The Engine of Context