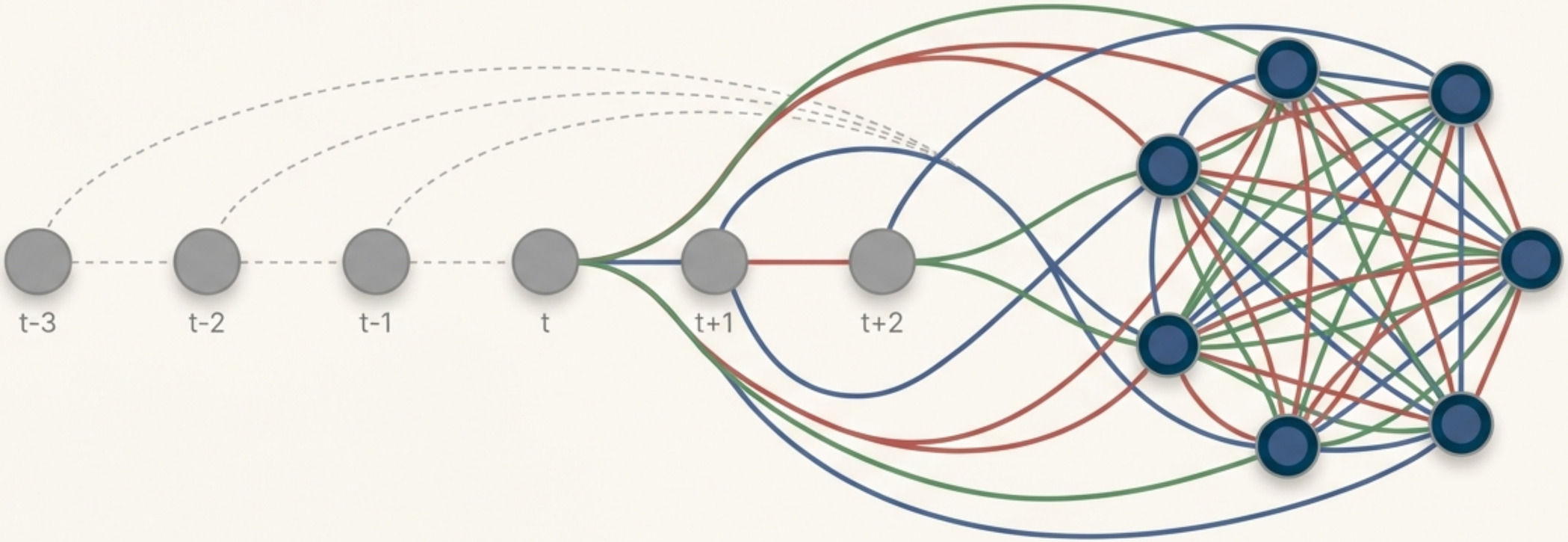
The Transformer: An Architectural Journey from Sequence to Self-Attention

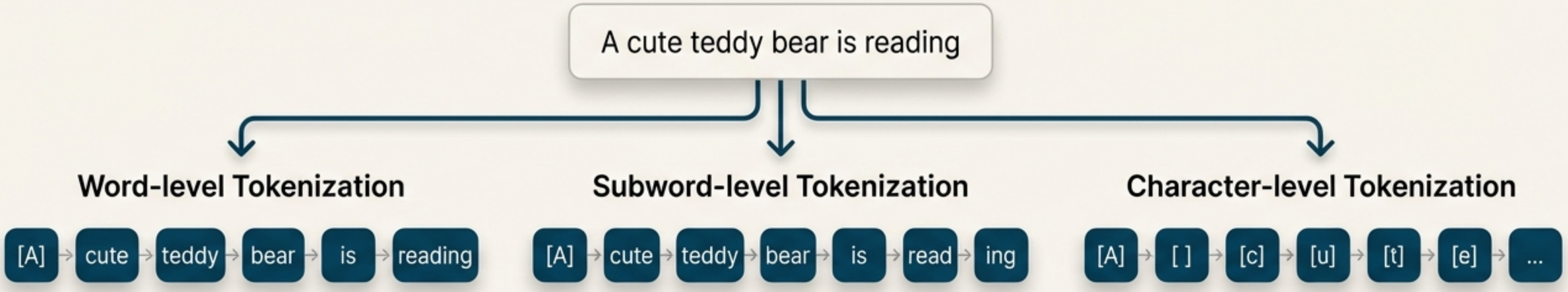
A guided deconstruction of the model that revolutionised natural language processing and powers modern Large Language Models.



Based on the Stanford CME295 lectures by Amin and Shervin Fath-Pour.

The First Hurdle: How do we translate words into guage machines understand?

The first step is **Tokenization**: breaking down text into smaller units called tokens. There are three main approaches, each with critical trade-offs.



Word-level Tokenization

The most intuitive approach, splitting text by spaces.

- Pro: Simple and direct.
- Con: Creates a massive vocabulary. Fails to see relationships (e.g., 'bear' vs 'bears'). High risk of **Out-of-Vocabulary (OOV)** tokens.

Subword-level Tokenization

The modern standard. Breaks words into common roots and affixes.

- Pro: Leverages word roots for a smaller, efficient vocabulary. Greatly reduces the OOV problem.
- Con: Results in longer token sequences, increasing computational cost.

Character-level Tokenization

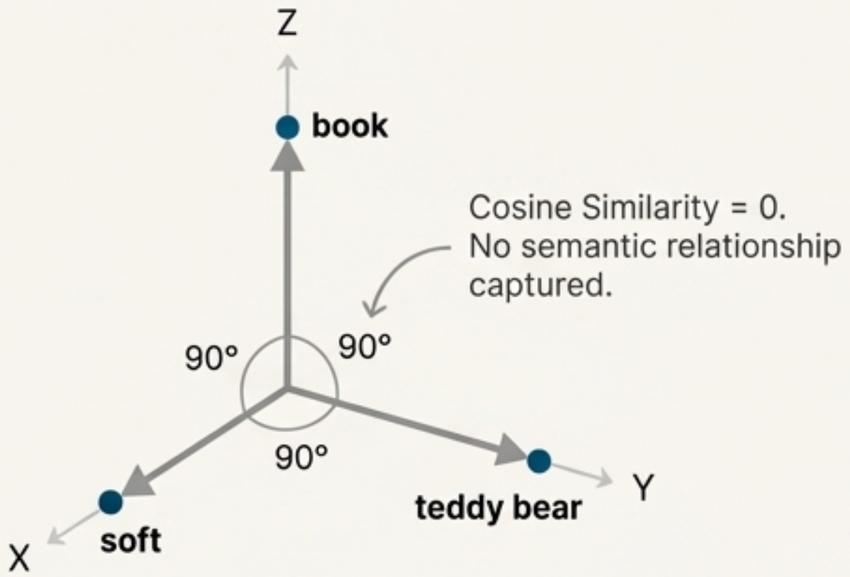
Splits text into individual characters.

- Pro: Robust to misspellings and casing errors. No OOV problem.
- Con: Creates extremely long sequences, making computation very slow. Meaning of a single character embedding is ambiguous.

From Tokens to Vectors: Learning the Meaning of Words

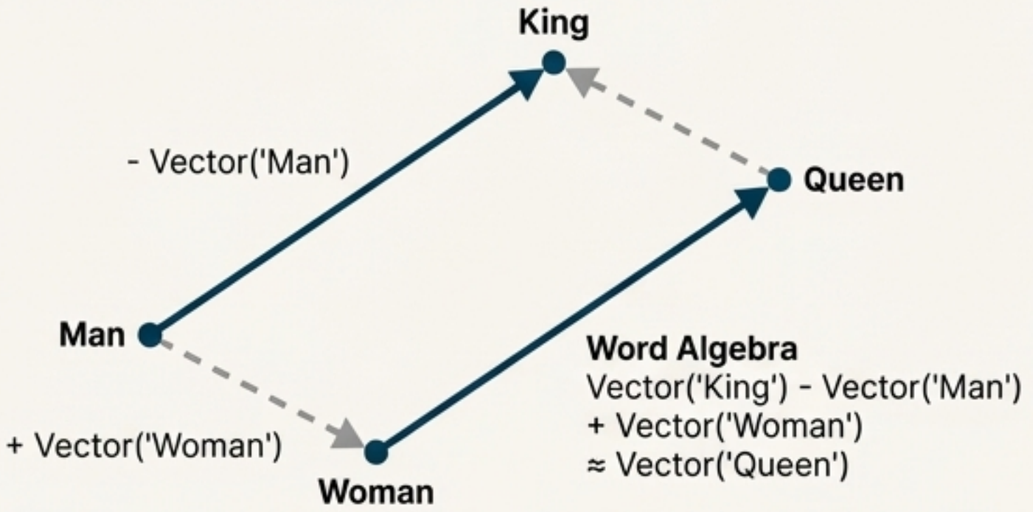
The Limitation of One-Hot Encoding

The naive approach assigns each token a unique vector (e.g., `[0, 0, 1, 0]`). The problem is that all token vectors are orthogonal, meaning their cosine similarity is zero. The model sees no relationship between 'soft' and 'teddy bear', which is semantically incorrect.



The Word2Vec Breakthrough

Instead of assigning vectors, we **learn** them from vast amounts of text using a **proxy task**, such as predicting a word from its context. The goal is not perfection on the proxy task, but to learn meaningful intermediate representations—the **embeddings**. The resulting dense vectors capture semantic relationships.



The Problem of Order: Context is More Than Just a Bag of Words

Core Problem: Methods like Word2Vec are context-independent. The word “bank” has the same vector in “robbing a bank” and “the river bank”. Word order is critical to meaning, but an average of word embeddings for a sentence loses all sequential information.

The Challenge: We need an architecture that processes text sequentially, building an understanding of the sentence as it reads, just like a human does.

Sentence A:
The dog chased the cat.

Sentence B:
The cat chased the dog.

Identical 'Bag of Words' Representation

`{The: 2, cat: 1, dog: 1, chased: 1}`

The Sequential Solution and Its Achilles' Heel

Recurrent Neural Networks (RNNs)

Concept: An architecture designed to handle sequences. It processes tokens one by one, maintaining a **hidden state** (or "context vector") that encapsulates the meaning of the sentence processed so far. This state is updated at each step.

Benefit: Word order now matters. The final hidden state is a representation of the entire sentence's meaning.

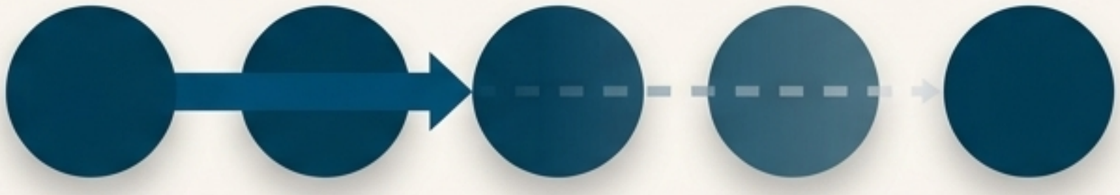
The Critical Flaw: Long-Range Dependencies

Problem: The hidden state acts as an information bottleneck. As the sequence gets longer, information from early tokens is diluted or lost. The model "forgets" the beginning of a long paragraph.

Technical Root Cause: The **vanishing gradient** problem. During training, the error signal weakens as it propagates back through time, making it impossible to update the model based on early inputs.

Note: LSTMs were developed to mitigate this, but still struggle with very long dependencies and remain inherently sequential and slow.

Sequential Processing



The Vanishing Gradient Problem

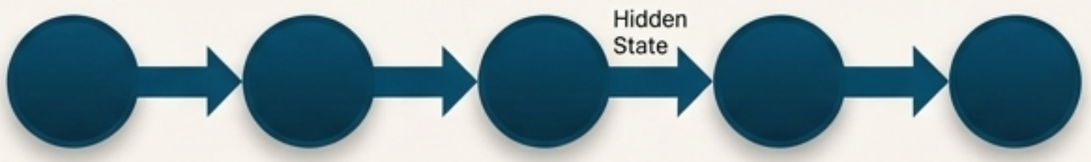
The Breakthrough: What if every word could talk to every other word at once?

Core Idea

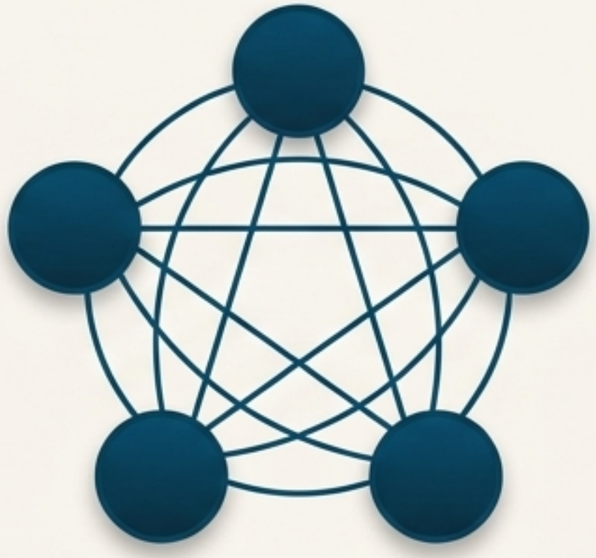
Instead of forcing information through a sequential bottleneck, the **Attention mechanism** creates direct connections between every token in the sequence. To compute the representation for one token, it can directly "look at" every other token and decide which are most relevant.

Implications

- **Solves Long-Range Dependencies:** The path between any two tokens is of length 1. No more vanishing gradients over sequence length.
- **Enables Parallelisation:** Computations for each token are independent and can be performed in parallel, a massive advantage for modern GPUs.



Sequential Processing

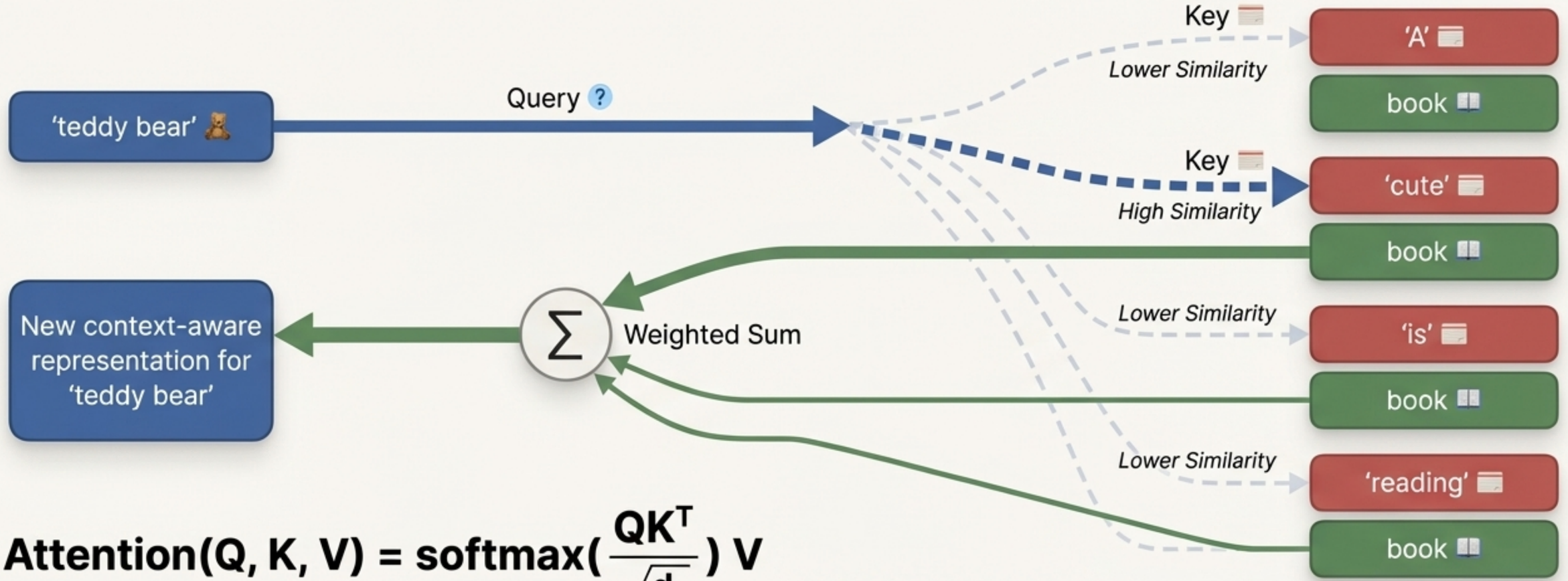


Direct Connections, Parallel Computation

Deconstructing Self-Attention: A Library Search Analogy

The attention mechanism operates using three learned vector representations for each input token:

- **Query (Q):** What I'm looking for. It's the current token asking, "Who in this sentence is relevant to me?"
- **Key (K):** The "index card" for every other token. It's what other tokens have to offer.
- **Value (V):** The actual content of the other tokens. It's the information to be retrieved.



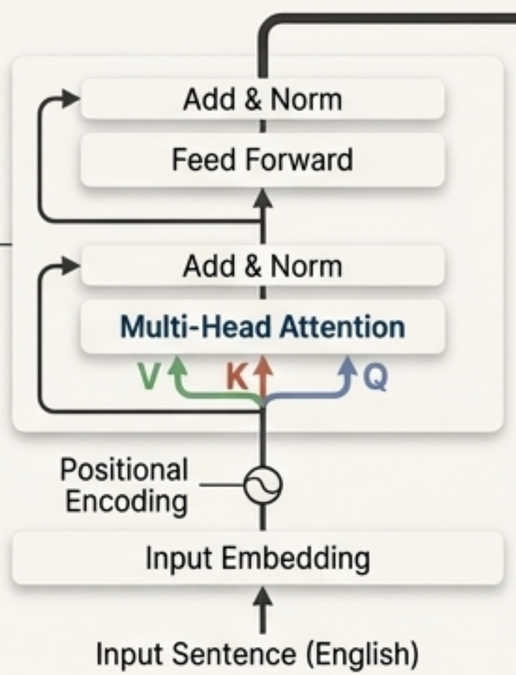
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V$$

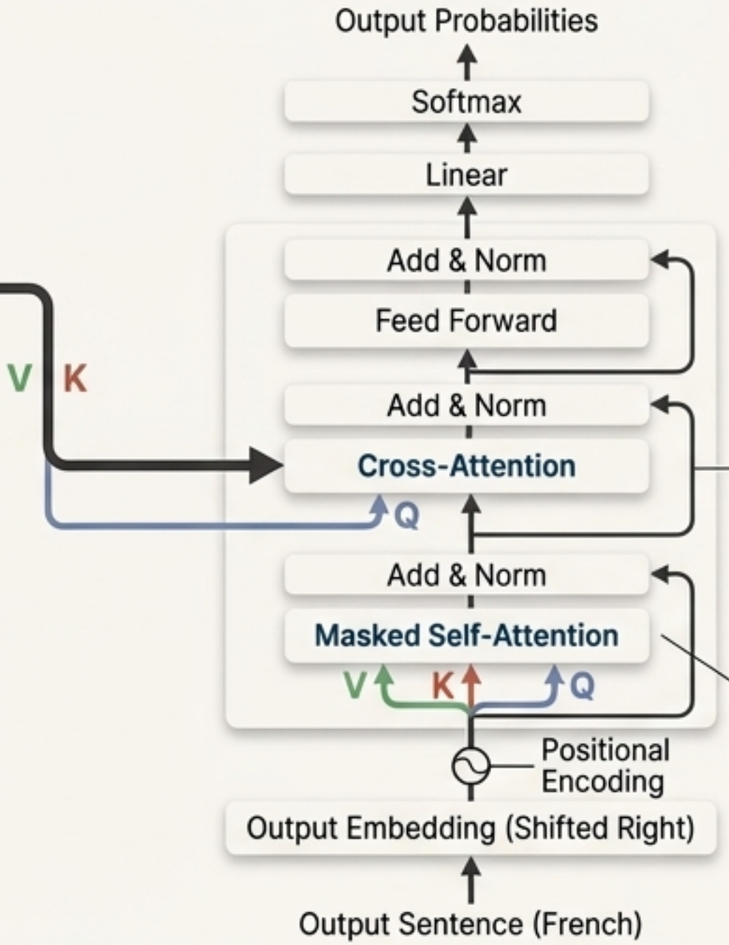
Scaling factor to stabilise gradients during training.

Assembling the Pieces: The Original Transformer Architecture

The Encoder

Processes the entire input sentence. Each token's representation is updated by attending to all other tokens in the input, building a rich, context-aware understanding.





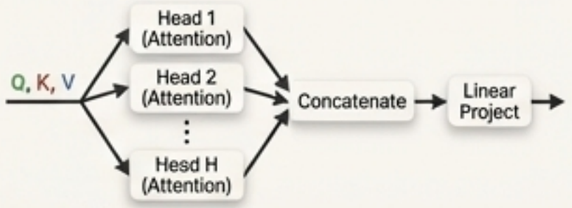
The Decoder

The decoder's queries attend to the keys and values from the final output of the Encoder. This is the crucial step where the decoder consults the input sentence.

Attends only to tokens it has already generated. Masked to prevent it from "cheating" by looking ahead.

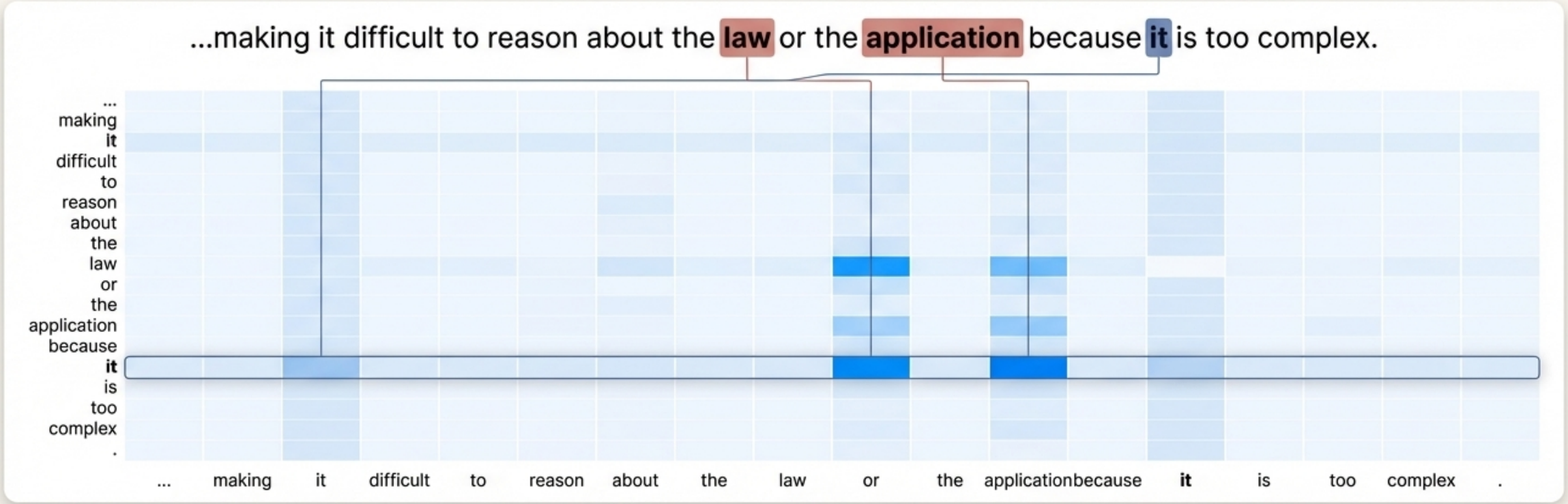
Multi-Head Attention

Instead of computing attention once, the model does it multiple times in parallel with different learned projections. Each "head" can learn to focus on different types of relationships (e.g., syntactic vs. semantic). The results are then combined.



Visualising Attention: How a Model ‘Sees’ Language

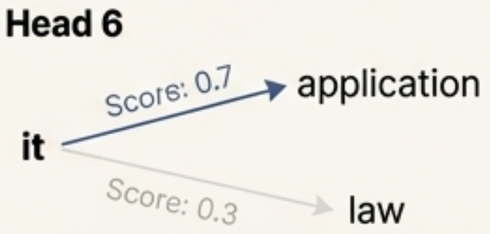
Core Concept: An **attention map** is a visualisation of the attention weights (the output of the softmax). It shows how much a query token attends to every key token in the sequence, revealing what the model has learned about linguistic structure.



Interpretation Section

The model correctly identifies the antecedents of the pronoun “it”.

Different attention heads learn different relationships. The visualisation below shows two heads: one focusing more on "law", the other more on "application”, demonstrating the power of the multi-head mechanism.



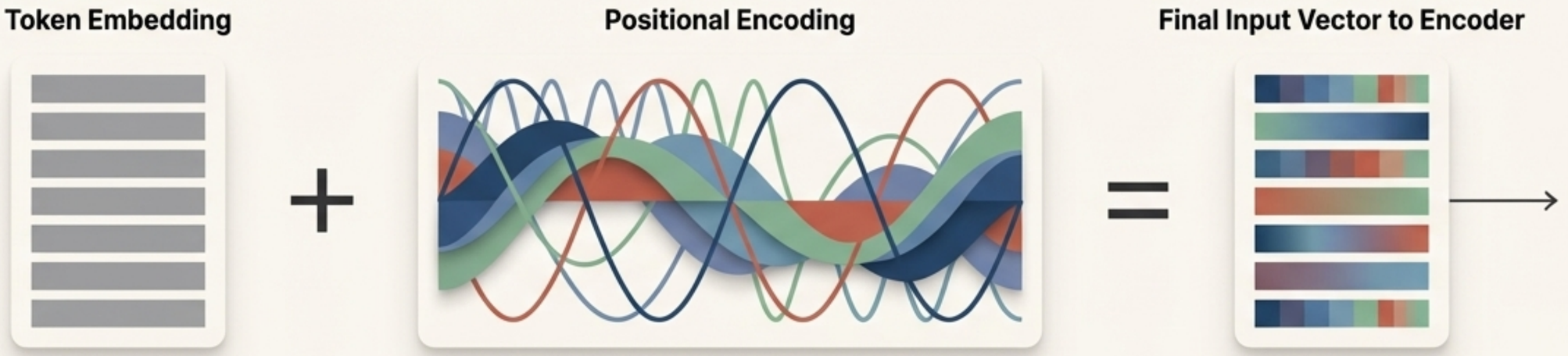
The Problem of Timelessness: Reintroducing Word Order

The Flaw

Self-attention, by its nature, is permutation-invariant. It treats the input as an unordered set of tokens. "The dog chased the cat" and "The cat chased the dog" would produce identical representations without extra information. We've lost the sequential information we had in RNNs.

The Original Solution: Positional Encodings

- **Concept**: We must explicitly inject information about a token's absolute position into its embedding *before* it enters the encoder stack.
- **Method**: A unique **positional encoding vector** is added to each token's input embedding.
- **Two Flavours**:
  1. **Learned Embeddings**: A unique, learnable vector for each position (pos 1, pos 2...). *Limitation*: Fails to generalise to sequences longer than those seen during training.
  2. **Sinusoidal Formula**: A fixed formula using sine and cosine functions of different frequencies to generate a unique vector for each position. *Benefit*: Can theoretically extend to any length.



Evolving Positionality: From Additive Encodings to Rotary Embeddings (RoPE)

The Problem with Additive Encodings

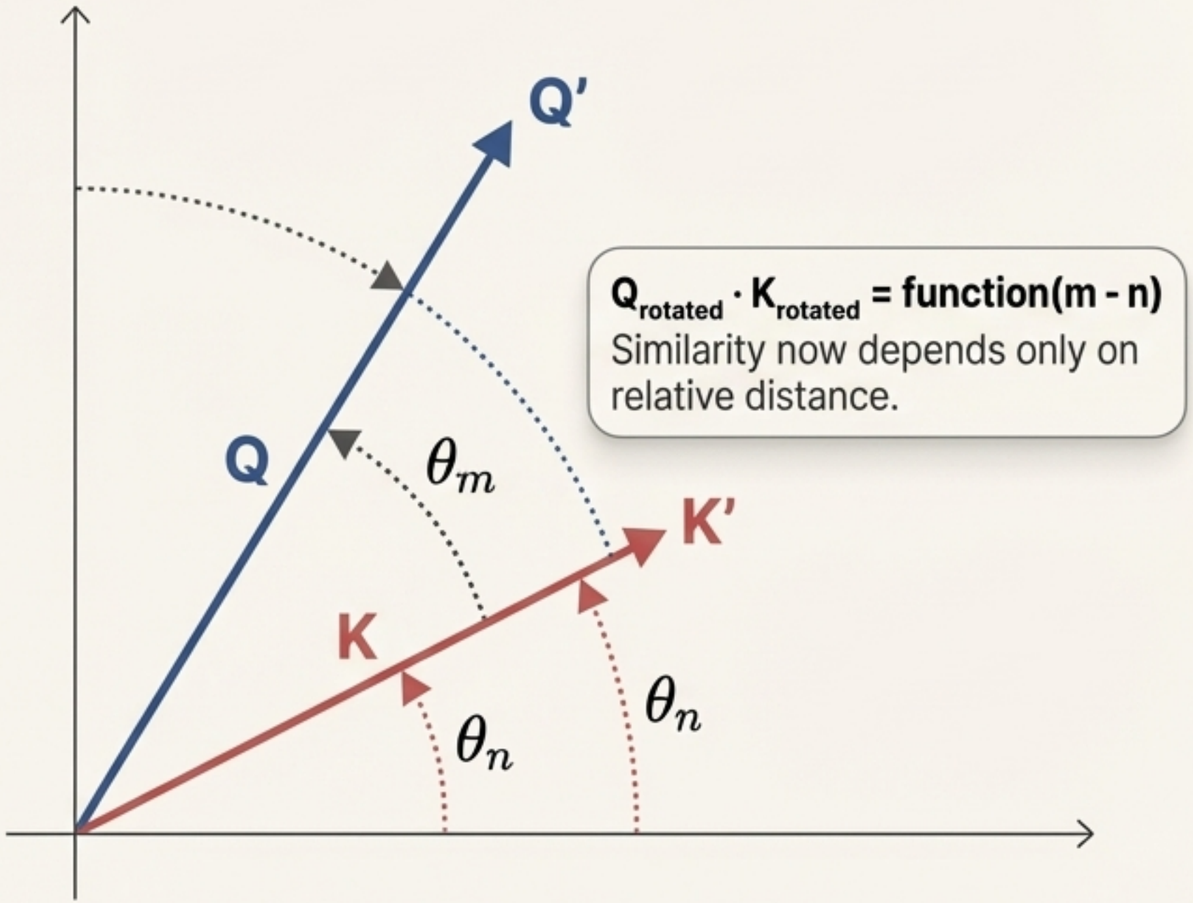
Injecting position information at the very beginning is indirect. We want position to matter most where similarity is calculated: inside the attention mechanism itself.

The RoPE Innovation

Concept: Instead of adding a vector, RoPE *rotates* the Query and Key vectors in multi-dimensional space. The angle of rotation is a function of the token's absolute position.

The Mathematical Elegance: The dot product between a Query rotated by angle `m` and a Key rotated by angle `n` is a function of only their *relative distance* (`m - n`). This elegantly encodes relative position directly into the similarity score.

Key Benefits: Naturally learns relative positions, attention scores decay with distance, and it extrapolates better to longer sequences.

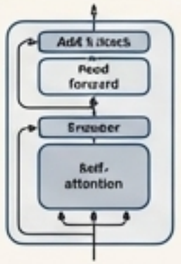


The Post-Transformer Cambrian Explosion: A Taxonomy of Architectures

The original 2017 architecture was just the beginning. The community quickly realised that its components could be unbundled and reconfigured into distinct families for different tasks.

Original Transformer (2017)

Encoder-Only



Models:
(e.g., BERT, RoBERTa)

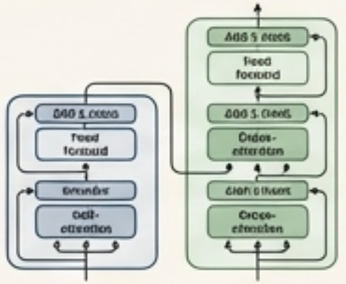
- **Description:** Uses only the Transformer's encoder stack.
- **How it Works:** Creates deep, bidirectional, context-aware representations of input text.
- **Primary Use Case:** Natural Language Understanding (NLU) tasks like sentiment analysis and classification. Excels at *analysis*, not generation.

Decoder-Only

Models:
(e.g., GPT series, Llama)

- **Description:** Uses only the Transformer's decoder stack (with masked self-attention).
- **How it Works:** Autoregressive next-token prediction. Excellent at generating fluent, coherent text.
- **Primary Use Case:** Large Language Models (LLMs), chatbots, summarisation, code generation.

Encoder-Decoder



Models:
(e.g., T5, BART)

- **Description:** Keeps the full original architecture.
- **How it Works:** Maps an input sequence to an output sequence.
- **Primary Use Case:** Sequence-to-sequence tasks like machine translation and text-to-text summarisation.

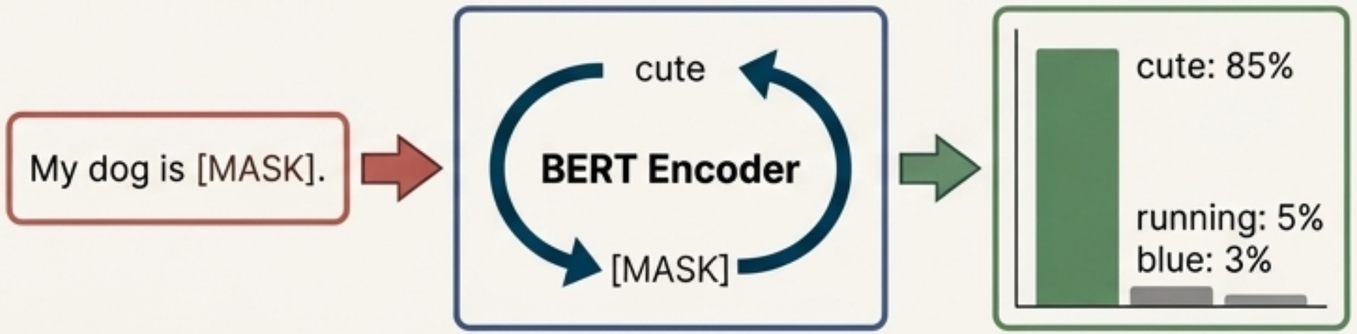
A Deep Dive into Analysis: The “Sesame Street” Era with BERT

BERT: Bidirectional Encoder Representations from Transformers.

- **Key Idea:** Pre-train a deep **bidirectional** model on a massive unlabelled text corpus, then fine-tune it on specific downstream tasks with very little labelled data.

Pre-training Objectives

1. **Masked Language Modelling (MLM):** 15% of input tokens are randomly masked. The model's goal is to predict the original identity of these masked tokens based on the full bidirectional context. This forces a deep understanding of language structure.
2. **Next Sentence Prediction (NSP):** Given two sentences, A and B, the model predicts if B is the actual sentence that follows A. (Note: This task was later found to be less helpful and was dropped by subsequent models like RoBERTa).



The 'Sesame Street' Connection

BERT's release followed another influential model named ELMo. This kicked off a trend of naming models after Sesame Street characters, becoming an inside joke in the research community.

Making Transformers Practical: Taming the O(n^2) Complexity

The Scaling Challenge

The core self-attention mechanism has computational and memory complexity that scales quadratically with sequence length n (O(n^2)). Doubling the context length quadruples the cost, making very long sequences infeasible.

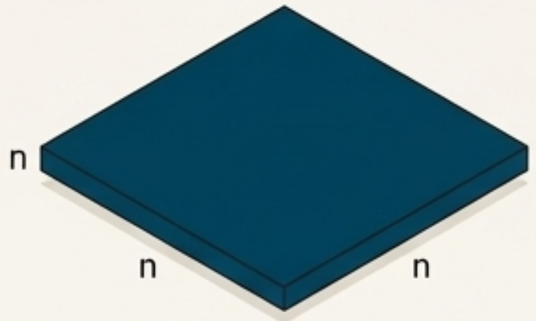
Key Optimisations

1. Sliding Window (Local) Attention (e.g., Longformer)

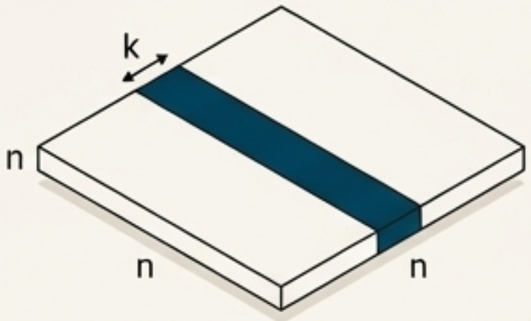
- **Concept:** Instead of full self-attention, each token only attends to a fixed-size window of neighbouring tokens (k to the left and k to the right).
- **Benefit:** Reduces complexity from O(n^2) to O(n*k), which is linear if k is fixed.

2. Grouped-Query & Multi-Query Attention (GQA & MQA)

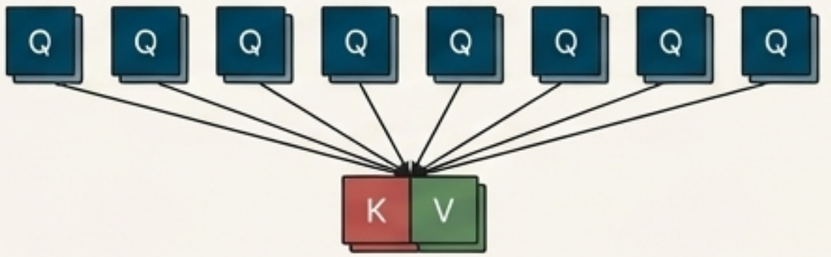
- **Problem:** During autoregressive decoding, the Keys (K) and Values (V) for all previous tokens must be stored in the 'KV Cache'. This cache becomes enormous in Multi-Head Attention.
- **Solution:** Share K and V projection matrices across multiple Q heads. MQA is the extreme (one K/V for all Qs), GQA is a hybrid.
- **Benefit:** Drastically reduces the KV cache size, saving memory and speeding up inference with minimal performance loss.



Full Attention - O(n^2)



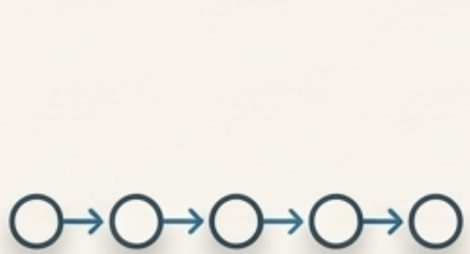
Sliding Window - O(n*k)

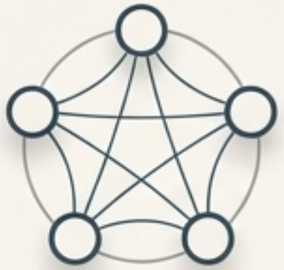


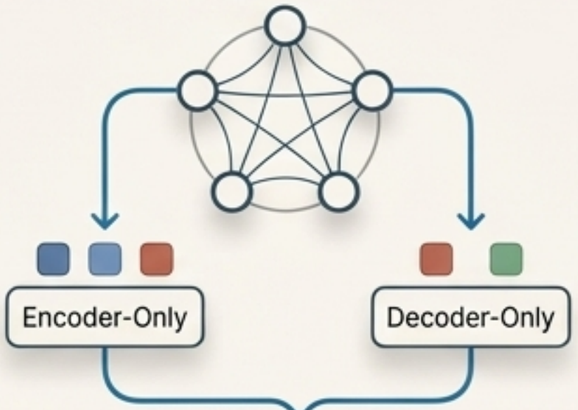
Multi-Query Attention - Reduced KV Cache

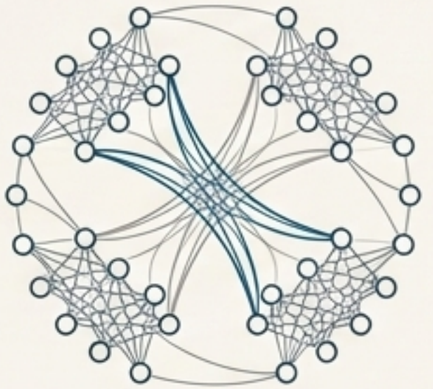
The Innovation Journey: From Sequential Chains to Parallel Webs of Meaning

The story of the Transformer is a story of identifying and breaking bottlenecks, evolving from **slow, sequential processing** to **vast, parallel networks of context**.









The Sequential Era (RNNs)

Innovation: Captured word order.

Limitation: The sequential bottleneck & vanishing gradient.

The Attention Breakthrough (Original Transformer)

Innovation: Direct connections (Self-Attention) solved long-range dependencies and enabled parallelism.

Principle: Attention is All You Need.

The Architectural Divergence (BERT, GPT)

Innovation: Specialised architectures (Encoder-Only for analysis, Decoder-Only for generation).

Principle: The right tool for the job.

The Era of Scale & Efficiency (RoPE, GQA)

Innovation: Sophisticated positional encodings and optimised attention mechanisms.

Principle: Making massive context windows computationally feasible.

The Transformer architecture was not an end-point, but a foundational primitive. Its core principles—**parallel computation** and **direct, context-aware connections**—continue to be the bedrock upon which the entire field of modern AI is built.