The Building Blocks of LLMs

A Mathematical History

From Archimedes to Attention:
2,000+ Years of Mathematics Inside Your AI

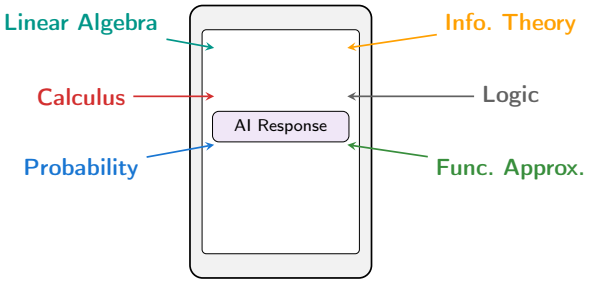
Prof. J. Osterrieder 2026

What's Inside Your AI?

When you type a question to ChatGPT or Claude...

The machine that answers is built entirely from mathematics.

- Not magic
- Not "intelligence"
- **Specific math**, invented by **specific people**, at **specific times**



Key Insight

You already know some of this math. You just don't know it's inside your AI yet.

The Exploded Transformer – What We Will Build Today

THE EXPLODED TRANSFORMER DIAGRAM (full slide, to be generated as figures/exploded-transformer-base.pdf). Components to show, color-coded by section: (1) Embedding layer – lookup table, word $\rightarrow$ vector [linalgTeal] (2) Positional encoding – sinusoidal waves added to embeddings [seqIndigo] (3) Multi-head attention block – Q/K/V projections, scaled dot-product, concat + linear [linalgTeal + probBlue] (4) Add & Norm (residual connection + layer normalization) [neuronPink] (5) Feed-forward network – two linear layers with activation [approxGreen + neuronPink] (6) Second Add & Norm [neuronPink] (7) Repeat $\times N$ layers indicator [logicGray] (8) Output linear projection + softmax [probBlue + infoAmber] (9) Loss computation – cross-entropy [infoAmber + calculusRed] (10) Backpropagation arrows flowing backward through all layers [calculusRed] Layout: vertical stack (input at bottom, output at top), each component a labeled rounded box. Intentionally overwhelming on first view.

This looks complicated now. By the end, you will know every piece — and WHO invented it.

Our Roadmap — The Convergence Diagram

CONVERGENCE DIAGRAM – INITIAL STATE (to be generated as figures/convergence-0.pdf). Layout: central large circle with "?" in gray. 10 satellite nodes in a semicircle (180° arc, evenly spaced), all in gray: (1) Linear Algebra [1809–1900s] (2) Calculus & Optimization [c. 250 BCE–1986] (3) Probability & Statistics [1654–1933] (4) Information Theory [1865–1948] (5) Logic & Computation [1854–1970s] (6) Function Approximation [1807–1991] (7) The Neuron & Depth [1943–2016] (8) Sequence Modeling [1906–2014] (9) The Transformer [2017] (10) Scaling to LLMs [2018–2026] Gray dashed arrows from each satellite toward center. All nodes dim/unlit. Target colors when lit: linalgTeal, calculusRed, probBlue, infoAmber, logicGray, approxGreen, neuronPink, seqIndigo, aiViolet, accentOrange.

Linear Algebra

The Skeleton of Neural Networks

1809–1900s

"Neural networks are, at their core, matrix multiplication machines."

What Is a Vector? — Forces, Velocities, and Words

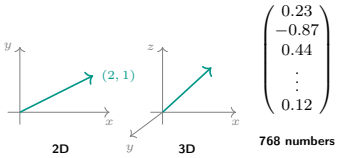
Hermann Grassmann (1809–1877)

Origin: Stettin, Prussia (now Szczecin, Poland)
Key work: *Die lineale Ausdehnungslehre* (1844) — the first general theory of vector spaces in any number of dimensions. A schoolteacher whose work was so ahead of its time that almost nobody read it.

Sir William Rowan Hamilton (1805–1865)

Origin: Dublin, Ireland
Key work: Invented quaternions (1843) while walking along the Royal Canal in Dublin — famously carved the formula into Brougham Bridge.

A **vector** is a list of numbers that represents something.



In an LLM, each word becomes a vector — a list of 768+ numbers.

Broom Bridge, Dublin

Here, on 16th October 1843, Sir William Rowan Hamilton carved the formula:

$$i^2 = j^2 = k^2 = ijk = -1$$

Modern Vector Notation — Gibbs and Heaviside

Josiah Willard Gibbs (1839–1903)

Origin: New Haven, Connecticut, USA
Key work: Extracted the useful vector parts from Hamilton's quaternions; created modern vector analysis (lecture notes *Elements of Vector Analysis*, c. 1881–1884).

Oliver Heaviside (1850–1925)

Origin: London, England
Key work: Self-taught electrical engineer who independently developed vector calculus; reformulated Maxwell's equations into their compact modern form.

Hamilton (1843)

$$\mathbf{q} = a + bi + cj + dk$$
$$\mathbf{q}\mathbf{q}' = \ldots$$

(12 terms)

simplify →

Gibbs (1881)

$$\vec{a} \cdot \vec{b} = a_1 b_1 + a_2 b_2 + a_3 b_3$$

(3 terms)

Same math. Simpler notation.

Matrices — Cayley and Sylvester

Arthur Cayley (1821–1895)

Origin: Richmond, Surrey, England
Key work: "A Memoir on the Theory of Matrices" (1858) — established matrices as algebraic objects in their own right.
Also a practicing lawyer for 14 years before returning to academia.

James Joseph Sylvester (1814–1897)

Origin: London, England
Key work: Coined the word "matrix" (1850), from Latin for "womb" — he saw it as something from which determinants are born. Also coined "discriminant," "invariant," and many other terms.

$$\begin{pmatrix} 2 & 0 & 1 \\ 3 & 1 & 4 \\ 7 & 5 & 2 \end{pmatrix}$$

Cayley, 1858

scale up

$$\begin{pmatrix} 0.02 & -0.7 & 0.31 & \cdots \\ 1.04 & 0.55 & \cdots & \\ \vdots & & \ddots & \end{pmatrix}$$

175 billion entries. Same math.

Matrix Multiplication — The Core Operation

Carl Friedrich Gauss (1777–1855)

Origin: Brunswick, Germany
Key work: Gaussian elimination (c. 1809–1810) for solving linear systems — laid groundwork for computational linear algebra. Originally used for astronomical orbit calculations.

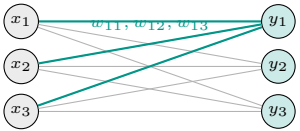
Key Insight

Every neural network layer is a matrix multiplication followed by a nonlinear function. That's it.

Matrix × vector:

$$\begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

Same operation as a layer:



Same math. Different picture.

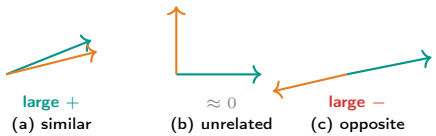
The Dot Product — Measuring Similarity

Recipe (the dot product in plain language):

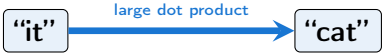
1. Take two lists of numbers, same length
2. Multiply matching entries:
   first $\times$ first, second $\times$ second, ...
3. Add up all the products
4. Result is **one number**:

Big positive	= very similar
Near zero	= unrelated
Big negative	= opposites

Formal: $\mathbf{a} \cdot \mathbf{b} = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$



The dot product measures how much two vectors agree.



*The model "attends" **it** to **cat**.*
This is how it resolves
what "it" refers to.

Cosine Similarity — Dot Product, Normalized

Recipe:

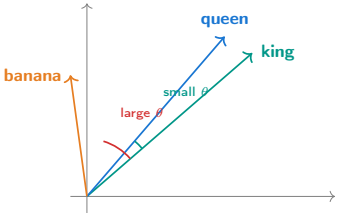
1. Compute the dot product
2. Divide by the length of each vector
3. Result is always between -1 and $+1$

Think of it as: *how much do these two vectors point the same way?* (Ignoring how long they are.)

Formal: $\cos(\theta) = \dfrac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}|\,|\mathbf{b}|}$

Key Insight

When you ask an LLM to find "similar words," it computes cosine similarity. The math is from the 1880s. The application is from last Tuesday.



Cosine similarity = the angle between word vectors.

Eigenvalues — The Natural Modes [EXTENDED]

Leonhard Euler (1707–1783)

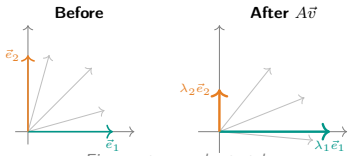
Origin: Basel, Switzerland / St. Petersburg / Berlin
Key work: First encountered eigenvalue-like concepts studying vibrating strings and rotating bodies (1740s–1750s).

Augustin-Louis Cauchy (1789–1857)

Origin: Paris, France
Key work: Proved symmetric matrices have real eigenvalues (1829).

David Hilbert (1862–1943)

Origin: Königsberg / Göttingen, Germany
Key work: Extended eigenvalues to infinite-dimensional spaces (spectral theory, early 1900s).



Eigenvectors: only stretch or shrink, never rotate.

LLM connection: Principal Component Analysis (PCA) uses eigenvalues to visualize and understand what LLMs learn. "What does this neuron represent?" — eigenvalue analysis helps answer.

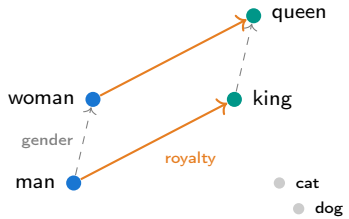
High-Dimensional Spaces — Where Words Live

LLMs represent words as vectors in spaces with **hundreds or thousands of dimensions**.

The math works the same as in 2D or 3D — we just can't draw it.

Richard Bellman (1920–1984)

Origin: New York City, USA
Key work: Coined "the curse of dimensionality" (c. 1957–1961): in high dimensions, data becomes sparse and distances behave counterintuitively.



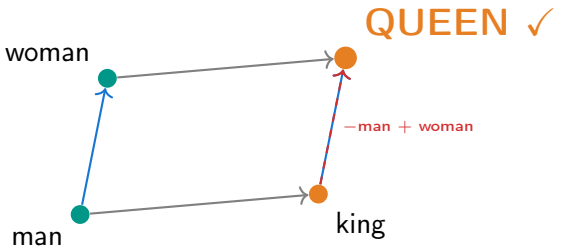
Projected from 768 dims to 2. Mikolov et al., 2013

Tomas Mikolov et al. demonstrated this with **Word2Vec** (2013) — word vectors trained on large text corpora capture semantic relationships as *linear directions*.

Quick Experiment — Word Vector Arithmetic

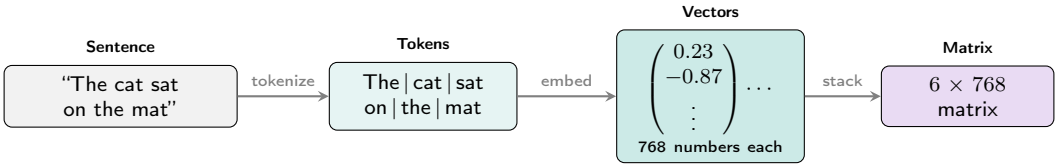
What does this equal?

$$\mathbf{v}_{\text{king}} - \mathbf{v}_{\text{man}} + \mathbf{v}_{\text{woman}} = ?$$



Mikolov et al., Word2Vec, 2013

From Words to Numbers — The Embedding



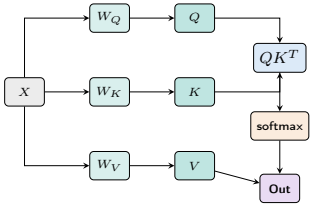
Your sentence is now a matrix. Everything that follows is linear algebra.

Historical lineage: Grassmann (1844, abstract vector spaces) → Mikolov (2013, Word2Vec learned embeddings) → Modern LLM embedding layers (2017–present, learned jointly with the transformer)

Attention as Matrix Multiplication — A Preview

The attention recipe (conceptual pipeline):

1. Start with input matrix X (sentence as matrix)
2. Multiply by $W_Q \rightarrow$ get Queries
   (matrix multiplication — Cayley)
3. Multiply by $W_K \rightarrow$ get Keys
   (matrix multiplication — Cayley)
4. Multiply by $W_V \rightarrow$ get Values
   (matrix multiplication — Cayley)
5. Scores = dot product of each Query with each Key
   (dot product — Gibbs, Slide 10)
6. Normalize scores with softmax
   (probability — Section 3)
7. Output = scores $\times$ Values



Formal (preview): $\text{Attn} = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$

K^T = "flip the matrix on its side" (transpose). $\sqrt{d}$ = scaling factor.
We will unpack this fully in Section 9.

Building the Diagram — Linear Algebra Complete

CONVERGENCE DIAGRAM – VARIANT 1

(figures/convergence-1.pdf). Node (1) "Linear Algebra" lit in linalgTeal with solid arrow to center. Label: "Matrix multiplication = neural network layers." Nodes (2)–(10) remain gray with dashed arrows. Central node still "?" in gray.

Summary — who built the skeleton:

1843	Hamilton — quaternions
1844	Grassmann — vector spaces
1850	Sylvester — named "matrix"
1858	Cayley — matrix theory
1880s	Gibbs/Heaviside — notation
1809+	Gauss — elimination
2013	Mikolov — Word2Vec

Key Insight

Every layer of an LLM is a matrix multiplication. Every attention score is a dot product. The math of 1850 is the engine of 2026.

Calculus & Optimization

How Neural Networks Learn

c. 250 BCE – 1986

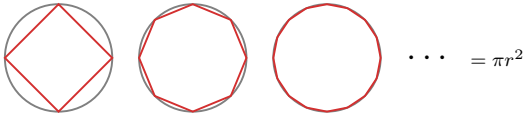
"A neural network learns by rolling downhill on a landscape made of calculus."

Roots in Antiquity — Archimedes and Infinitesimals

Archimedes of Syracuse

Origin: Syracuse, Sicily
Key work: The "method of exhaustion" — calculated areas and volumes by approximating with ever-finer polygons.

His lost *Method* manuscript, rediscovered in 1906 in the Archimedes Palimpsest, shows he used infinitesimal reasoning more explicitly than previously known.



Approximate, refine, repeat.
This is what calculus does.

Key Insight

The core idea of calculus: approximate with simple pieces, take the limit. Archimedes had this 1,900 years before Newton.

The Derivative — Measuring Change

Pierre de Fermat (1601–1665)

Origin: Beaumont-de-Lomagne, France
Key work: Method for tangent lines (c. 1636–1638); found maxima and minima by setting a difference quotient to zero.

Isaac Newton (1642–1726/27)

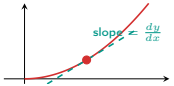
Origin: Woolsthorpe, England
Key work: "Method of fluxions" (c. 1665–1666) — rates of change for physics.

G. W. Leibniz (1646–1716)

Origin: Leipzig, Germany
Key work: Independently developed calculus (published 1684–1686) with the dy/dx notation we still use today.

Fermat		Newton		Leibniz
$\frac{f(x+e)-f(x)}{e}$, $e \to 0$	=	$\dot{y}$, fluxion	=	$\frac{dy}{dx}$, still used

Three notations, one idea: rate of change.



The derivative = the slope of this line.

LLM connection: Training means computing derivatives of the error w.r.t. *every* weight — billions of weights, billions of derivatives, millions of times.

The Chain Rule — The Key to Backpropagation

Recipe (the chain rule in plain language):

If y depends on u and u depends on x...

To find how much x affected y through the chain, multiply the effects at each step.

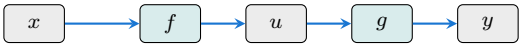
Formal: $\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}$

Leibniz's notation makes this look almost obvious — the du's "cancel." (They don't really, but the intuition helps.)

Key Insight

Backpropagation uses the chain rule to figure out which weights to blame for the error.

Math:

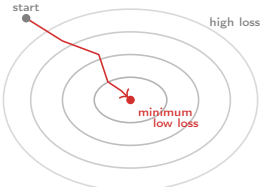


NN:

Gradient Descent — Rolling Downhill [EXTENDED]

Augustin-Louis Cauchy (1789–1857)

Origin: Paris, France
Key work: Published the first description of the method of steepest descent (1847), *Méthode générale pour la résolution des systèmes d'équations simultanées*.



Gradient descent: always walk downhill.
Cauchy, 1847.

Recipe:

1. Compute the gradient ("which way is uphill?")
2. Take a small step in the opposite direction
3. Repeat

Cauchy was solving equations, not training neural networks.
But the algorithm is the same.

Backpropagation — The Algorithm That Changed Everything

Seppo Linnainmaa (b. 1945)

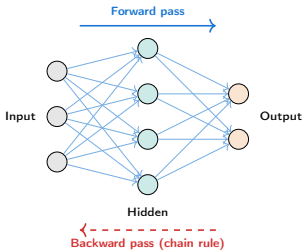
Origin: Finland
Key work: Published automatic differentiation in his Master's thesis (1970, University of Helsinki) — the mathematical core of backpropagation.

Rumelhart/Hinton/Williams (1986)

David Rumelhart (1942–2011, USA)
Geoffrey Hinton (b. 1947, London, UK/Canada)
Ronald Williams (USA)

Published "Learning representations by back-propagating errors" (1986, *Nature*). Demonstrated backprop could train multi-layer networks. Reignited the field.

Hinton shared the 2024 Nobel Prize in Physics with John Hopfield for foundational work enabling machine learning.



Timeline:

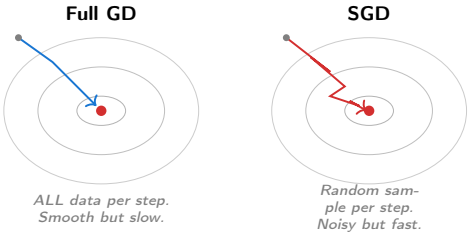
1847 Cauchy — steepest descent
1970 Linnainmaa — auto. differentiation
1986 Rumelhart/Hinton/Williams — backprop
2024 Hinton — Nobel Prize in Physics

Stochastic Gradient Descent — Learning from Samples

Robbins & Monro (1951)

Herbert Robbins (1915–2001, New Castle, PA, USA)
Sutton Monro (1920–1995, USA)

Published the Robbins–Monro algorithm (1951) — the first stochastic approximation method. The theoretical foundation for SGD.



Key Insight

Every LLM you have ever used was trained with some variant of SGD. The Robbins–Monro paper from 1951 is running inside every AI lab today.

The Loss Landscape — A Visual Tour

3D LOSS LANDSCAPE (to be generated as figures/loss-landscape.pdf via Python/matplotlib or pgfplots standalone). Style: Li et al., 2018 "Visualizing the Loss Landscape of Neural Nets." Surface: 3D plot of $f(x, y) = 0.5(x^2 + y^2) + 3\exp(-2((x-1)^2 + (y+1)^2)) - \exp(-3((x+1.5)^2 + (y-0.5)^2))$ *or similar multi-modal surface. Labeled annotations: (A) "Global minimum" – deepest valley, green marker (B) "Local minimum" – secondary valley, yellow marker (C) "Saddle point" – flat ridge, gray marker (D) "SGD path" – noisy red trajectory from random start to global min. Colormap: viridis or coolwarm. View angle: 30° elevation.*

This is what training an LLM looks like mathematically. The landscape has *billions* of dimensions — we can only visualize a 2D slice.

Building the Diagram — Calculus Complete

CONVERGENCE DIAGRAM – VARIANT 2

(figures/convergence-2.pdf). Node (1) "Linear Algebra" lit in linalgTeal (from Section 1). Node (2) "Calculus & Optimization" now lit in calculusRed. Label: "Gradient descent = how it learns." Nodes (3)–(10) remain gray with dashed arrows. Central node still "?" in gray.

Summary — who taught it to learn:

c. 250 BCE	Archimedes — proto-calculus
c. 1636	Fermat — tangents, extrema
1665–66	Newton — fluxions
1684–86	Leibniz — dy/dx notation
1847	Cauchy — steepest descent
1951	Robbins–Monro — SGD
1970	Linnainmaa — auto. diff.
1986	Rumelhart/Hinton/Williams — backpropagation

Key Insight

Gradient descent + backpropagation = the learning algorithm of every LLM.

Session 1 Complete

We know the skeleton and how it learns. Next: how does it handle uncertainty?

Probability & Statistics
The Language of Uncertainty

1654–1933

Pascal · Fermat · Bernoulli · Bayes · Kolmogorov · Fisher · Shannon · Boltzmann

The Birth of Probability — A Gambling Problem (1654)

Blaise Pascal (1623–1662)

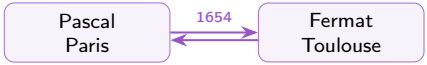
Origin: Clermont-Ferrand, France

Mathematician, physicist, philosopher. Co-founded probability theory in correspondence with Fermat (1654) about the problem of points: how to divide stakes in an interrupted game of chance.

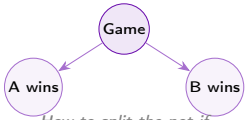
Pierre de Fermat (1601–1665)

Origin: Beaumont-de-Lomagne, France

His reply to Pascal laid out the counting arguments that became **combinatorial probability**.



Problem of Points



How to split the pot if the game stops early?

From Gambling to Laws — Bernoulli and Laplace

Jacob Bernoulli (1655–1705)

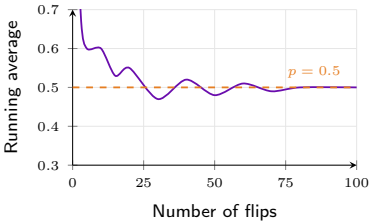
Origin: Basel, Switzerland

Proved the law of large numbers (published posthumously in *Ars Conjectandi*, 1713): as you repeat an experiment, the observed frequency converges to the true probability.

Pierre-Simon Laplace (1749–1827)

Origin: Beaumont-en-Auge, France

Théorie analytique des probabilités (1812) systematized the entire field. Popularized Bayes' theorem.



Key Insight

Why do LLMs improve with more data? Bernoulli's law: more data $\Rightarrow$ learned probabilities converge to the true patterns of language.

Bayes' Theorem — Updating Beliefs with Evidence (1763)

Thomas Bayes (c. 1701–1761)

Origin: London, England

Presbyterian minister and mathematician. *Essay towards solving a Problem in the Doctrine of Chances* published **posthumously** (1763) by his friend Richard Price.

Recipe: Updating Beliefs

New belief = $\frac{\text{(likelihood of evidence if guess right)} \times \text{(prior guess)}}{\text{(likelihood of evidence overall)}}$

Formal

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)}$$

Surprise Example

Test accuracy: 99%
Disease prevalence: 1 in 10,000
Positive test ⇒ chance you have it?

Answer: ∼1%
Most positives are *false* positives!

Key Insight

LLMs implicitly learn conditional probabilities:
$P(\text{next word} \mid \text{previous words})$

Kolmogorov's Axioms — Probability Made Rigorous (1933)

Andrey Kolmogorov (1903–1987)

Origin: Tambov, Russia

Published *Grundbegriffe der Wahrscheinlichkeitsrechnung* (Foundations of the Theory of Probability, 1933), axiomatizing probability using measure theory.

This put probability on the same rigorous footing as geometry (Euclid) and calculus (Cauchy/Weierstrass).

[EXTENDED] — can be cut without losing the narrative

Kolmogorov's Three Axioms

1. **Non-negativity:** Probabilities are between 0 and 1.
2. **Certainty:** Something must happen (total probability = 1).
3. **Additivity:** Probabilities of mutually exclusive events add up.

Three simple rules. Every probability calculation in every LLM obeys them.

Maximum Likelihood — Finding the Best Fit

Ronald A. Fisher (1890–1962)

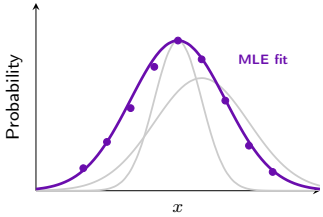
Origin: London, England

Introduced maximum likelihood estimation (concept 1912, formalized 1922). His 1922 paper "On the Mathematical Foundations of Theoretical Statistics" established MLE as a foundational method.

Recipe: Maximum Likelihood

Given the data I observed, which parameters make this data **most probable**?

$\rightarrow$ Choose the best fit.



Key Insight

LLM training is MLE at scale: find weights that maximize the probability of the training text. Fisher's method from 1922, applied to trillions of words.

Statistical Language Models — Shannon's Breakthrough (1948)

Claude Shannon (1916–2001)

Origin: Petoskey, Michigan, USA

Published *A Mathematical Theory of Communication* (1948), founding information theory. Modeled English as a stochastic process and built **n-gram language models** decades before AI.

Shannon (1948): predict next letter using previous N letters

↓

LLM (2024): predict next token using ALL previous tokens

Shannon's Approximations to English

0th order (random letters):
`XFOML RXKHRJFFJUJ`

1st order (letter frequencies):
`OCRO HLI RGWR NMIE`

2nd order (letter pairs):
`ON IE ANTSOUTINYS`

Word-level bigram:
`THE HEAD AND IN FRONTAL ATTACK`

More context ⇒ more readable!

Key Insight

Shannon was doing language modeling in 1948 — 70 years before GPT. The same core idea, vastly more data.

The Softmax Function — From Physics to Token Prediction

Ludwig Boltzmann (1844–1906)

Origin: Vienna, Austria

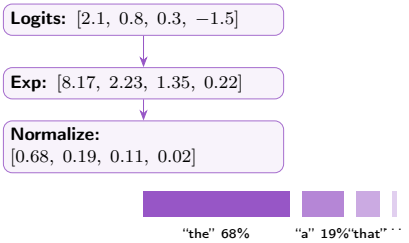
His Boltzmann distribution (1870s) describes the probability of a physical system being in a state based on its energy: $P \propto e^{-E/kT}$.

Lineage: Boltzmann (1870s, physics) → Gibbs (1900s, statistical mechanics) → Luce (1959, psychology of choice) → **Softmax** (ML)

Recipe: Softmax

1. Take each raw score, raise e to that power
(makes everything positive, magnifies differences)
2. Add up all the results
3. Divide each by the total

⇒ Now they are probabilities that sum to 1!

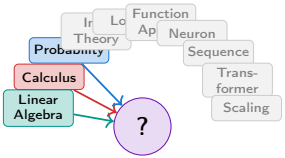


Formal

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

$\sum$ = "add up for all j"

Building the Transformer — Probability & Statistics



Softmax = next-token prediction

Section 3 Summary

Who / When	Contribution
Pascal / Fermat (1654)	Birth of probability
Bernoulli (1713)	Law of large numbers
Bayes (1763) / Laplace	Updating beliefs from evidence
Kolmogorov (1933)	Rigorous axioms
Fisher (1922)	Maximum likelihood estimation
Shannon (1948)	Language as a statistical process
Boltzmann (1870s)	$\rightarrow$ Luce (1959) $\rightarrow$ Softmax

Key Insight

Every time an LLM generates a word, it computes a softmax probability distribution. The math started with a gambling question in 1654.

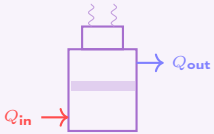
Information Theory
Measuring Meaning

1865–1948

Clausius · Boltzmann · Shannon · Kullback · Zipf

Entropy — Invented Three Times

Clausius (1865)



Entropy as "energy unavailable for work." Named from Greek *entropia* (transformation).

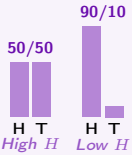
Boltzmann (1870s)

Tombstone: $S = k \log W$

Entropy as "number of arrangements."

$$S = k \log W$$

Shannon (1948)



Entropy as "average surprise."

$$H = -\sum_i p_i \log p_i$$

$\sum$ = "add up for every outcome i"

The same formula. Three completely different domains.

Cross-Entropy — THE Loss Function of LLMs

Recipe: Cross-Entropy Loss

1. The true answer is one specific word.
2. Look at the probability your model gave that word.
3. Take the logarithm.
4. Negate it.

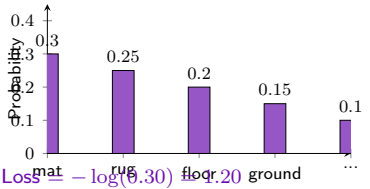
High confidence in right answer ⇒ low loss.
Low confidence in right answer ⇒ high loss.

Formal

$$\text{Loss} = -\log\big(p_{\text{model}}(\text{correct token})\big)$$

Full form uses $\sum$ over all tokens, but since only one is correct, it simplifies to this.

"The cat sat on the _____"



Loss $= -\log(0.30) = 1.20$

If model improves to 80% for "mat": loss $= -\log(0.80) = 0.22$

Key Insight

Training an LLM = minimizing cross-entropy = making the model more confident about the RIGHT next word.

Surprisal and Perplexity — How We Evaluate LLMs

Surprisal

Surprisal $= -\log\big(p(\text{word})\big)$

High probability $\Rightarrow$ low surprisal
Low probability $\Rightarrow$ high surprisal

Perplexity

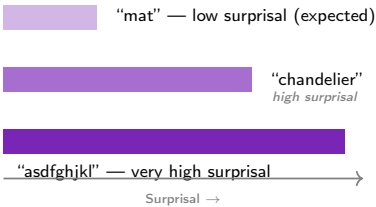
Average surprisal, exponentiated.

Lower perplexity = better model.

"GPT-4 has lower perplexity than GPT-3" means it's less surprised by real text.

Key Insight

A good language model is rarely surprised by real text. Shannon would have understood this immediately.



KL Divergence — Measuring the Gap Between Distributions

Solomon Kullback (1907–1994) & **Richard Leibler** (1914–2003), USA.
Published KL divergence in 1951.

Recipe: KL Divergence

For each possible outcome, ask:
"How much more surprising is this under model Q compared to truth P?"

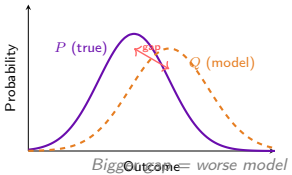
Average over all outcomes.

If Q matches P perfectly $\Rightarrow$ KL $= 0$.

Key Insight

In RLHF (the technique that makes ChatGPT helpful), KL divergence prevents the model from changing too much during fine-tuning.

[EXTENDED] — can be cut without losing the narrative



Formal

$$D_{\mathrm{KL}}(P\|Q) = \sum_i P(i)\,\log\frac{P(i)}{Q(i)}$$

Tokenization — Carving Language into Pieces

George Kingsley Zipf (1902–1950, USA). Zipf's law (1935/1949): word frequency is inversely proportional to its rank. "The" is #1, "of" is #2, ...

Rico Sennrich, **Barry Haddow**, **Alexandra Birch** (Edinburgh, 2016).
Adapted Byte Pair Encoding (BPE) for neural machine translation. BPE is the tokenization method used by GPT and most modern LLMs.

Key Insight

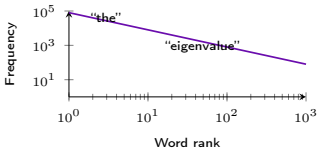
The AI doesn't see words — it sees tokens. BPE gives short tokens to common patterns (Zipf's law!) and splits rare words into pieces.

BPE Tokenization Example

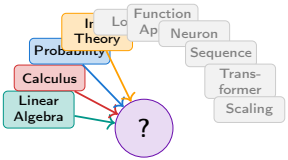
Input:
`The mathematician calculated the eigenvalue`

Tokens:
`The` `mathematic` `ian` `calculated` `the` `eigen` `value`

Common words → single token
Rare words → split into pieces



Building the Transformer — Information Theory



Cross-entropy = the loss function

Section 4 Summary

Who / When	Contribution
Clausius (1865)	Thermodynamic entropy
Boltzmann (1870s)	Statistical entropy, $S = k \log W$
Shannon (1948)	Information entropy, the "bit"
Kullback–Leibler (1951)	Divergence between distributions
Zipf (1935/1949)	Word frequency laws
Sennrich et al. (2016)	BPE tokenization

Key Insight

LLMs are trained by minimizing cross-entropy loss. Shannon's 1948 formula is literally the objective function.

Logic & Computation
The Substrate

1854–1970s

Boole · Frege · Gödel · Turing · von Neumann

"Logicians dreamed of mechanical thought.
They proved it was impossible.
Then neural networks did it anyway."

Boolean Algebra — The Language of Computers (1854)

George Boole (1815–1864)

Origin: Lincoln, England; worked at Queen's College Cork, Ireland

Published *An Investigation of the Laws of Thought* (1854). Self-taught mathematician who showed that **logical reasoning** could be expressed as **algebra**: variables take values 0 (false) or 1 (true).

Key Insight

Boole was a self-taught son of a shoemaker who became a university professor. His algebra is now the foundation of every digital device on Earth.

AND

A	B	Out
0	0	0
0	1	0
1	0	0
1	1	1

OR

A	B	Out
0	0	0
0	1	1
1	0	1
1	1	1

NOT

A	Out
0	1
1	0

Every GPU in every AI data center is made of *billions* of these gates.

The Dream and Its Limits — From Frege to Turing

After Boole, logicians dreamed of reducing ALL reasoning to mechanical symbol manipulation. They almost succeeded — and then proved it was **impossible**.

Frege (1879)
First formal logical system.
A precursor to programming languages.

→

Russell & Whitehead
Principia (1910–13)
362 pages to prove $1+1=2$.

→

Gödel (1931)
Incompleteness: true statements exist that *cannot* be proved.

→

Turing (1936)
Halting problem: no algorithm can decide if any program will stop.

Key Insight

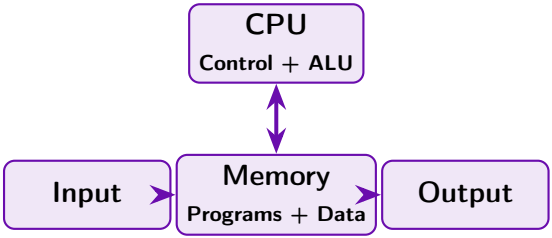
The irony: neural networks, built from Boole's logic gates, learned to translate, reason, and create — not by following rules, but by learning patterns from data. The machines that logic built went beyond what logic can do.

Von Neumann Architecture — The Hardware Blueprint (1945)

John von Neumann (1903–1957)

Born: Neumann János Lajos
Origin: Budapest, Hungary; worked at Princeton, USA

Published *First Draft of a Report on the EDVAC* (1945), describing the stored-program architecture: instructions and data share the same memory.



This basic design has powered nearly every computer for 80 years.

[EXTENDED] — can be cut; summarize verbally

LLM inference runs on this architecture. But training requires massively parallel hardware. Enter the GPU.

Floating Point — How Computers Handle Real Numbers

Konrad Zuse (1910–1995, Berlin, Germany). Built the Z3 (1941), the first programmable computer using floating-point arithmetic.

How 3.14159 is stored

IEEE 754 standard (1985)

Key Insight

Every weight in an LLM is a floating-point number. Large models have hundreds of billions of them.

Precision vs. Speed Trade-off

Format	Bits	Use
FP32	32	Traditional training
FP16 / BF16	16	Modern LLM training
INT8	8	Fast inference
INT4	4	Aggressive compression

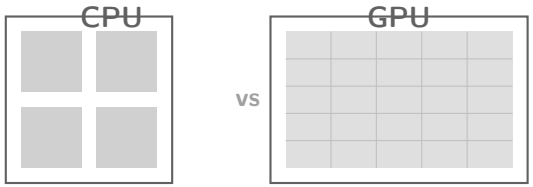
Fewer bits = less precise but faster and smaller. Modern LLMs accept small rounding errors in exchange for fitting bigger models.

From Zuse's Z3 (1941) to today's GPU clusters: the same fundamental idea, vastly more scale.

GPU Computing — Why Matrix Multiplication Won

NVIDIA released CUDA (2007): general-purpose computing on GPUs.

Alex Krizhevsky, **Ilya Sutskever**, **Geoffrey Hinton** — AlexNet (2012): GPUs could train deep neural networks dramatically faster, launching the deep learning era.



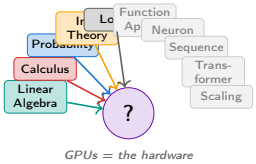
Key Insight

Without GPUs, deep learning would still be a curiosity. Video game hardware accidentally enabled AI.

"AlexNet showed GPUs could train networks 10–100× faster than CPUs. That single result changed the trajectory of AI."

Neural network training: millions of *independent* matrix multiplications $\Rightarrow$ perfect for GPU.

Building the Transformer — Logic & Computation



GPUs = the hardware

Key Insight

Boolean logic built the hardware. Turing defined computation. GPUs made deep learning possible. And neural networks surpassed what formal logic can do.

Section 5 Summary

Who / When	Contribution
Boole (1854)	Boolean algebra
Frege (1879)	Formal logic
Russell/Whitehead (1910–13)	*Principia Mathematica*
Gödel (1931)	Incompleteness theorems
Turing (1936)	Computability, Turing machine
Von Neumann (1945)	Stored-program architecture
Zuse (1941)	Floating-point computation
CUDA (2007) + AlexNet (2012)	GPU revolution

Session 2 Complete

"We have the math and the hardware. But can we actually build a brain?"

Function Approximation

Why Neural Networks Work at All

1807–1991

Fourier · Weierstrass · Cybenko · Hornik

A neural network can approximate *any* function.
Here is why that is remarkable.

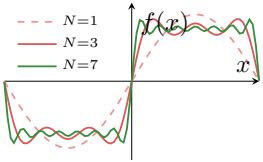
Joseph Fourier — Any Function as a Sum of Waves

Jean-Baptiste Joseph Fourier (1768–1830)

Origin: Auxerre, France

Key work: *Théorie analytique de la chaleur* (1822, based on work from 1807). To solve the heat equation, Fourier claimed that any function could be decomposed into sines and cosines.

Lagrange objected. The claim was initially controversial — and proved enormously fertile.



Any function = a sum of simple waves. *Fourier, 1807.*

Key Insight

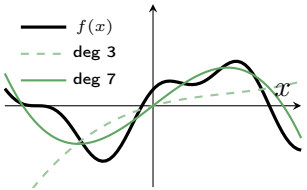
LLM preview: Positional encoding in the transformer uses sine and cosine functions at different frequencies — directly inspired by Fourier's idea. We will see this in Section 9.

Karl Weierstrass — Polynomials Can Do It Too [EXTENDED]

Karl Weierstrass (1815–1897)

Origin: Ostenfelde, Westphalia, Germany

Key result (1885): For any continuous function f on a closed interval and any desired accuracy $\varepsilon > 0$, there exists a polynomial P that stays within ε of f everywhere on that interval.



Any continuous function can be approximated as closely as desired by a polynomial.

Key Insight

Polynomials are **universal approximators** for continuous functions. A century later, Cybenko would prove the same thing for *neural networks*.

The Universal Approximation Theorem

George Cybenko (b. c. 1952)

Origin: Canada/USA

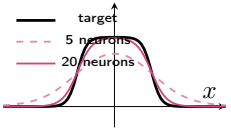
Key result (1989): A feedforward network with a single hidden layer of sigmoidal neurons can approximate *any* continuous function to any desired accuracy.

"Approximation by Superpositions of a Sigmoidal Function"

Kurt Hornik (b. c. 1963)

Origin: Austria

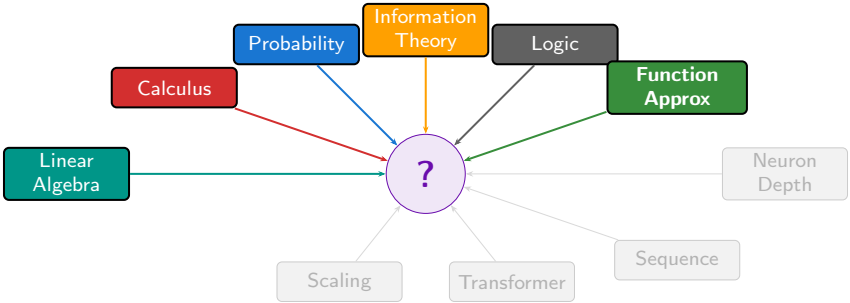
Key result (1991): Generalized Cybenko's theorem to a wider class of activation functions. The universality is not about sigmoid — it is about the *architecture*.



More neurons = better approximation.
Cybenko, 1989.

A neural network **can** learn any continuous function, given enough neurons. This is not an analogy. It is a *mathematical theorem*.

Convergence Diagram — Function Approximation Lights Up



Universal approximation = why it works. **6 of 10 building blocks active.**

Section 6 Summary

Fourier (1807/1822)	Any function as waves
Weierstrass (1885)	Polynomial approximation
Cybenko (1989)	Neural net universality
Hornik (1991)	Generalized activation

→ Connection to the Transformer

The universal approximation theorem guarantees that neural networks can represent any continuous function. The LLM's ability to model language is a specific case.

The Neuron & Depth

From Biology to Deep Networks

1943–2016

McCulloch & Pitts · Rosenblatt · Minsky · Hinton · He

A real neuron fires or does not. A mathematical neuron does the same thing, with an equation. But stacking them deep required more invention.

McCulloch & Pitts — The First Mathematical Brain Cell

Warren McCulloch (1898–1969)

Origin: Orange, New Jersey, USA

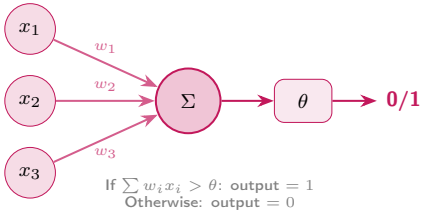
Neurophysiologist who sought to understand the brain in logical terms.

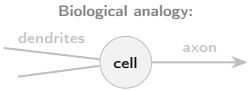
Walter Pitts (1923–1969)

Origin: Detroit, Michigan, USA

Self-taught logician. A troubled prodigy who ran away from home and ended up working with McCulloch.

Together they published *"A Logical Calculus of the Ideas Immanent in Nervous Activity"* (1943).





Frank Rosenblatt — The Perceptron: A Machine That Learns

Frank Rosenblatt (1928–1971)

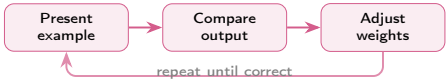
Origin: New Rochelle, New York, USA

Built the **Mark I Perceptron** (1957–1958) at the Cornell Aeronautical Laboratory. A physical machine (not a simulation) that could learn to classify images.

The *New York Times* reported it as a machine that could "walk, talk, see, write, reproduce itself, and be conscious of its existence."

Mark I Perceptron (1958)

A room-sized machine with 400 photocells connected to artificial neurons. Weights adjusted by electric motors. It could learn to recognise simple visual patterns.



Key Insight

The perceptron learning algorithm is gradient descent on a single neuron. Rosenblatt's machine could *learn*. The hype was enormous.

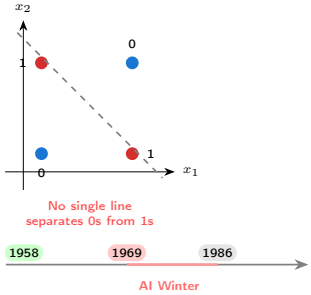
The XOR Problem — The Crash That Froze AI

Minsky (1927–2016) & Papert (1928–2016)

Marvin Minsky: New York City, USA
Seymour Papert: Pretoria, South Africa / USA

Published *Perceptrons* (1969): a rigorous mathematical analysis proving that a single-layer perceptron cannot learn XOR.

The math was correct. But the field interpreted it as: neural networks are a dead end.



The AI Winter (∼1969–1986)

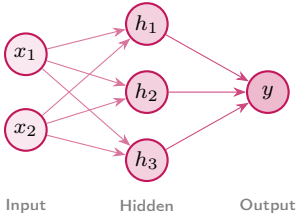
Funding dried up. Neural network research became unfashionable. Nearly two decades of stagnation. *The solution was right there: add more layers.*

Multi-Layer Networks — The Comeback

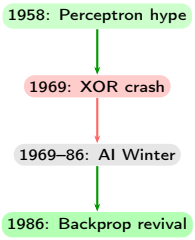
1986: The Revival

Rumelhart, Hinton & Williams (Slide 22) demonstrated that backpropagation could train multi-layer networks.

Multi-layer perceptrons CAN learn XOR — and much more. A hidden layer creates a *nonlinear* decision boundary.



Two layers **CAN** separate XOR.



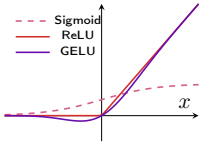
Key Insight

It took **17 years** for the field to recover from *Perceptrons*. The AI winter was caused by stopping **one layer too soon**.

Activation Functions — The Spice That Makes It Work

Key Insight

Without a nonlinear activation function, a multi-layer network collapses to a single layer. Activation functions give neural networks their power.



These curves make neural networks *nonlinear* — and therefore powerful.

Sigmoid — $\dfrac{1}{1+e^{-x}}$

- Smooth S-curve, used 1980s–2000s
- Related to the logistic function of **Pierre François Verhulst** (1804–1849, Brussels), who modelled population growth (1838)

ReLU — $\max(0, x)$

- Popularised by **Nair & Hinton** (2010)
- Simpler, faster, avoids vanishing gradients

GELU — used in GPT and BERT

- Introduced by **Hendrycks & Gimpel** (2016)
- Smooth version of ReLU; used in modern transformers

Residual Connections — The Trick That Enabled Depth

Kaiming He et al. (b. c. 1984)

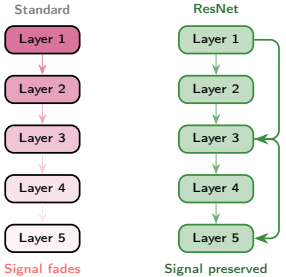
Origin: China / USA

Key paper (2015): *"Deep Residual Learning for Image Recognition"* (ResNet).

Instead of each layer computing a completely new output, it computes a change to the input:

$$\text{output} = \text{input} + \text{layer}(\text{input})$$

This simple idea allowed networks to grow from ∼20 layers to 152 layers — and beyond.



Key Insight

The "**Add**" in the transformer's "Add & Norm" block IS He's residual connection from 2015. Without it, transformers could not be deep.

Layer Normalization & Dropout — Stabilising Deep Networks

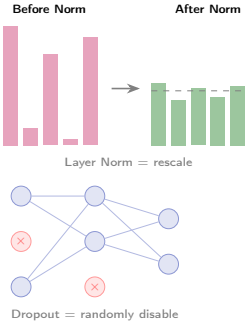
Ba/Kiros/Hinton – Layer Norm (2016)

Adjusts the outputs of each layer so they have a consistent mean and variance. Stabilises training and speeds convergence.

Srivastava/Hinton et al. – Dropout (2014)

Randomly deactivates neurons during training, forcing the network to learn robust features rather than memorising.

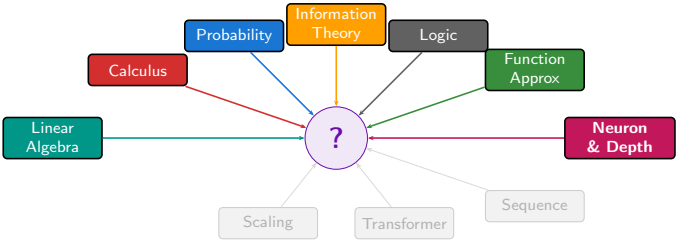
Like studying for an exam by covering random notes — you learn the material, not the page layout.



Layer Norm recipe: (1) Compute the average of all neuron outputs in a layer. (2) Compute how spread out they are. (3) Subtract the average and divide by the spread. Now they are centred at zero with consistent scale.

In the transformer's "Add & Norm":
Add = He's residual connection (Slide 60)
Norm = Ba's layer normalisation

Convergence Diagram — The Neuron & Depth Lights Up



Activation + normalisation + residuals = deep networks. **7 of 10 building blocks active.**

Section 7 Summary

McCulloch-Pitts (1943)	First math neuron
Rosenblatt (1958)	Perceptron learns
Minsky/Papert (1969)	XOR, AI winter
Rumelhart/Hinton/W. (1986)	Multi-layer revival
Verhulst (1838) → Sigmoid	Population → activation
Nair/Hinton (2010)	ReLU
Srivastava et al. (2014)	Dropout
He et al. (2015)	Residual connections
Ba et al. (2016)	Layer normalisation
Hendrycks/Gimpel (2016)	GELU

→ Connection to the Transformer

The artificial neuron, with its activation function, is the basic unit. Residual connections, layer normalisation, and dropout allow transformers to be DEEP. There are billions of neurons in GPT, stacked 96+ layers.

Sequence Modeling

The Path to Language

1906–2014

Markov · Baum · Elman · Hochreiter & Schmidhuber · Bahdanau

Language is a sequence. Modeling sequences required 100 years of invention.

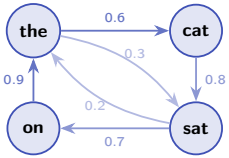
Andrey Markov — The Simplest Language Model

Andrey Andreyevich Markov (1856–1922)

Origin: Ryazan, Russia

Introduced **Markov chains** (1906) by analysing the alternation of vowels and consonants in Pushkin's poem *Eugene Onegin*.

He was *not* studying language — he was disproving a claim that the law of large numbers required independence. But his method became the foundation of all sequence modelling.



Markov: "the cat sat on the cat sat on the…"
LLM: "The cat sat on the warm mat, purring softly."

Key Insight

A Markov chain predicts the next state based *only* on the current state, ignoring all history. Your phone's autocomplete is a descendant of this idea.

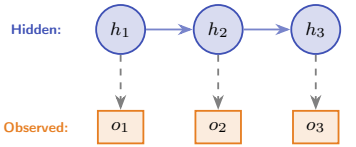
Hidden Markov Models — Adding Depth

Leonard Baum et al. (1960s–1972)

Origin: USA

Developed the mathematical framework for **Hidden Markov Models** (HMMs) and the **Baum–Welch algorithm** for training them.

HMMs add a hidden state layer: the observed output depends on an unobserved internal state that must be inferred.



Hidden states inferred from observations

Key Insight

Speech recognition used HMMs for 30+ years before deep learning replaced them. The idea of "hidden internal states" producing observable outputs is conceptually similar to the hidden layers of a neural network.

Recurrent Neural Networks — Memory in a Loop

Michael Jordan – Jordan Network (1986)

Origin: USA

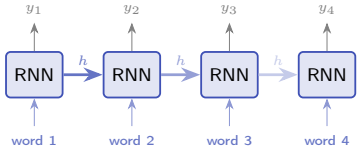
An early form of recurrent neural network.
Yes, the field has its own Michael Jordan.

Jeffrey Elman (1948–2018) — Elman Network (1990)

Origin: USA

The hidden state at time t is fed back as input at time $t+1$. This is the standard form of RNN.

Fatal flaw: The hidden state gets overwritten at each step. After 50 words, the memory of word 1 has been washed away. This is the vanishing gradient problem.



Hidden state = memory. But it fades fast.

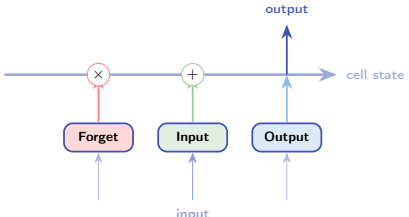
LSTM — Long Short-Term Memory [EXTENDED]

Hochreiter & Schmidhuber

Sepp Hochreiter: Austria
Jürgen Schmidhuber: Munich, Germany

Published *"Long Short-Term Memory"* (1997).

Hochreiter's diploma thesis (1991, supervised by Schmidhuber) identified the vanishing gradient problem. The 1997 LSTM paper provided the solution: explicit **memory cells** with **gates** that control what to remember and what to forget.



Key Insight

LSTM dominated language tasks from the late 1990s through 2017. Google Translate used LSTMs. But LSTMs process words *one at a time* — you cannot parallelise them. This bottleneck led to attention.

The Attention Mechanism — The Bridge to Transformers

Bahdanau/Cho/Bengio (2014)

Dzmitry Bahdanau: Belarus/Canada
Kyunghyun Cho: South Korea/Canada
Yoshua Bengio: France/Canada

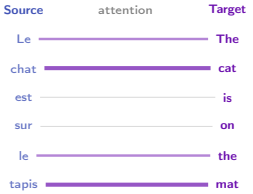
Published *"Neural Machine Translation by Jointly Learning to Align and Translate"* (2014 arXiv; 2015 ICLR).

Key insight: Instead of compressing the entire input into a fixed-size vector, let the model look back at all previous positions and decide which ones are relevant.

The Breakthrough

Attention = the ability to look back and focus.

Without this 2014 paper, there would be no GPT, no Claude, no ChatGPT.



RNN: compress all into one vector → bottleneck
Attention: look back at *everything* → no bottleneck

The Transformer

Where Everything Converges

2017

Vaswani · Shazeer · Parmar · Uszkoreit · Jones
Gomez · Kaiser · Polosukhin

Eight branches of mathematics meet in one architecture.

"Attention Is All You Need."

“Attention Is All You Need” — The Paper That Changed AI

In 2017, a team at Google published a paper proposing an architecture that used **only attention** — no recurrence, no convolution. It was simpler, faster, and better.

arXiv:1706.03762 (2017)

Attention Is All You Need

Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser, Polosukhin

NeurIPS 2017

The Eight Authors (2017 NeurIPS)

Ashish Vaswani, **Noam Shazeer**, **Niki Parmar**, **Jakob Uszkoreit**, **Llion Jones**, **Aidan Gomez**, **Łukasz Kaiser**, **Illia Polosukhin**

Most in their 20s and 30s. Several have since founded major AI companies (Cohere, Character.AI).

Eight researchers. One paper. One architecture.

Key Insight

Eight researchers. One paper. The architecture that powers *every* major LLM.

Self-Attention — The Core Mechanism

Each token generates three vectors: a **Query** ("what am I looking for?"), a **Key** ("what do I contain?"), and a **Value** ("what information do I carry?").

Self-Attention Step by Step

1. Compute attention scores: **dot product** of each Query with each Key (Gibbs, Slide 10)
2. Scale scores: divide by $\sqrt{d}$ to keep numbers manageable (cosine-like scaling, Slide 11)
3. Normalise with **softmax**: scores become probabilities (Boltzmann, Slide 33)
4. Output = weighted sum of Values via **matrix multiplication** (Cayley, Slide 9)

Formal notation (small inset):

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

$K^\top$ means "flip the matrix on its side" (transpose).

Concrete example: In "The cat sat on the mat because *it* was tired," the word "it" attends most strongly to "cat." Self-attention learns that "it" refers to "cat."

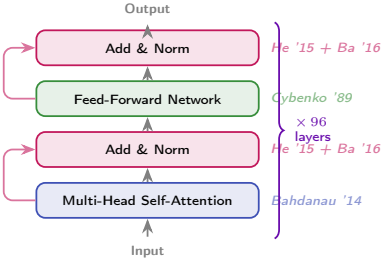
Multi-Head Attention & Feed-Forward Networks

Multi-Head Attention

Instead of one set of Q/K/V, the transformer uses multiple "heads" (8, 12, or more) in parallel. Each head learns different relationships: syntax, semantics, position, etc. Results are concatenated and projected back.

Feed-Forward Network (FFN)

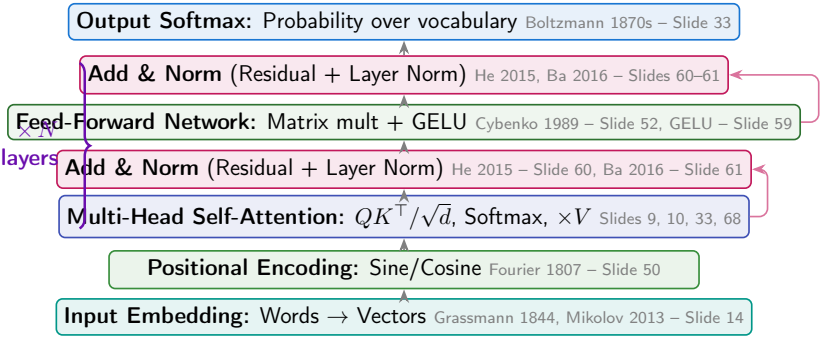
Two matrix multiplications with a GELU activation in between. This is the universal approximation theorem (Cybenko, Slide 52) applied within each transformer layer — the FFN provides the raw computational power.



Key Insight

One transformer layer = Attention + FFN + two rounds of "Add & Norm." Stack 96 of these and you have GPT-4.

The Full Transformer — Annotated with 2,000+ Years of Math



Training: Cross-entropy (Shannon '48) + Backprop (Linnainmaa '70) + SGD (Cauchy 1847) **Hardware:** GPU (Slide 47)

Interactive Moment: "Count the Building Blocks"

Students: look at this diagram and try to identify which of the 10 sections each component comes from. How many can you find? *(3 min)*

Positional Encoding — Fourier Meets Transformers

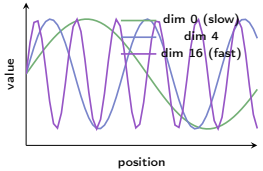
Transformers have no built-in notion of word order (unlike RNNs). To fix this, they add **positional encodings** — sine and cosine functions at different frequencies. This is Fourier's idea, applied directly.

Recipe (CONCEPTUAL):
For each position in the sentence and each dimension in the vector, compute a sine or cosine wave with a specific frequency. Different dimensions use different frequencies — just like Fourier's decomposition. Each position gets a unique "fingerprint" of wave values.

Formal:

$$PE(\text{pos}, 2i) = \sin(\text{pos} / 10000^{2i/d})$$
$$PE(\text{pos}, 2i{+}1) = \cos(\text{pos} / 10000^{2i/d})$$

The $10000^{2i/d}$ creates a spectrum of frequencies from very slow to very fast.



Fourier's sine waves tell the transformer which word is which. Each position gets a unique "fingerprint" of wave values.

Key Insight

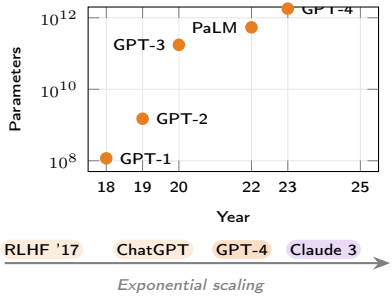
Fourier decomposed heat into waves. Vaswani decomposed word position into waves. Same math, 210 years apart.

The Scaling Leap — From Transformer to LLM

The transformer architecture, scaled up with more parameters, more data, and more compute, produces Large Language Models with emergent abilities.

The GPT Series

GPT-1 (2018)	117M params	Pre-train + fine-tune
GPT-2 (2019)	1.5B params	Convincing text gen.
GPT-3 (2020)	175B params	Few-shot learning
GPT-4 (2023)	undisclosed	Multimodal
Claude (2023–26)	—	Constitutional AI



Jared Kaplan et al. (2020) — Scaling Laws

LLM performance scales as a smooth **power law** with compute, data, and parameters. Predictable improvement.

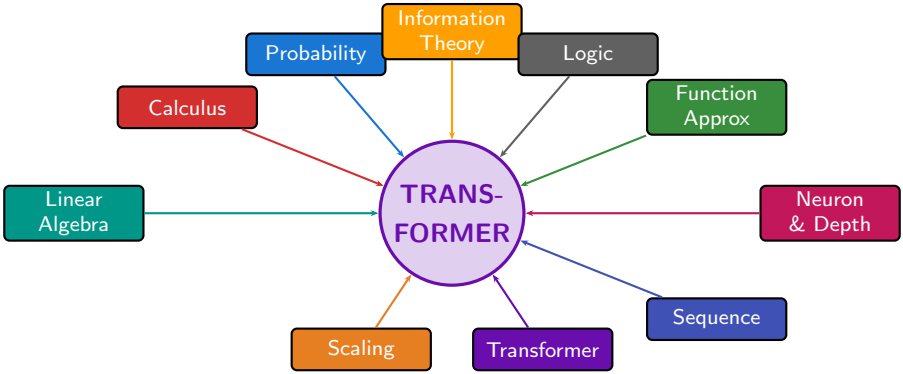
Return to the Exploded Diagram — Now You Understand Every Piece

Every Piece — Now You Know the History

Embedding (Slide 14)	Grassmann 1844, Mikolov 2013
Positional Encoding (Slide 50)	Fourier 1807 — sine/cosine waves
Self-Attention (Slides 68, 71)	Bahdanau 2014, Vaswani 2017
Dot product (Slide 10)	Gibbs, 1880s
Softmax (Slide 33)	Boltzmann, 1870s
Matrix mult (Slide 9)	Cayley, 1858
Add & Norm (Slides 60–61)	He 2015 (residual) + Ba 2016 (layer norm)
FFN + GELU (Slides 52, 59)	Cybenko 1989, Hendrycks 2016
Output Softmax (Slide 33)	Boltzmann/Luce
Training (Slides 21, 22, 37)	Cauchy 1847 (GD), Linnainmaa 1970, Shannon 1948
Hardware (Slide 47)	GPU computing, NVIDIA CUDA 2007

You now know the history of every piece of this machine.

Convergence Diagram — Final State

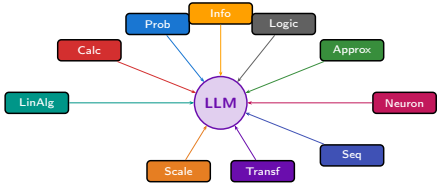


Ten streams of human invention, flowing into one machine.

The Story Is Not Over

∞ 2,000+ Years of Mathematics. One Machine. Your Turn.

- Every component of an LLM was invented by someone, for a specific purpose, often centuries before AI existed.
- Grassmann imagined abstract vector spaces. Boltzmann studied gas molecules. Fourier solved the heat equation. Markov analysed poetry. Archimedes approximated circles. **None of them imagined AI.**
- The math you learn in school — algebra, calculus, probability — is the *same math* running inside the AI you talk to every day.
- The next building block has not been invented yet. It might come from pure mathematics, from physics, from linguistics, from neuroscience — **or from someone in this room.**



2,000+ years of mathematics. One machine. Your turn.