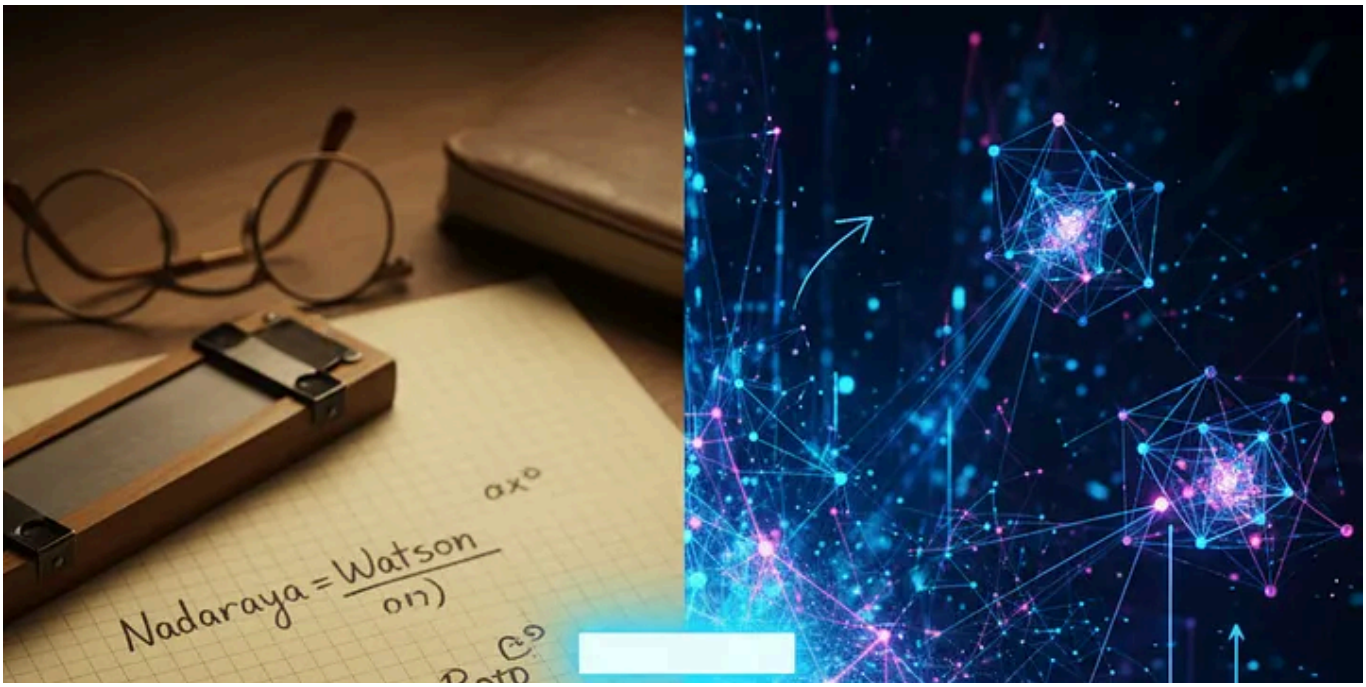
Level Up Coding

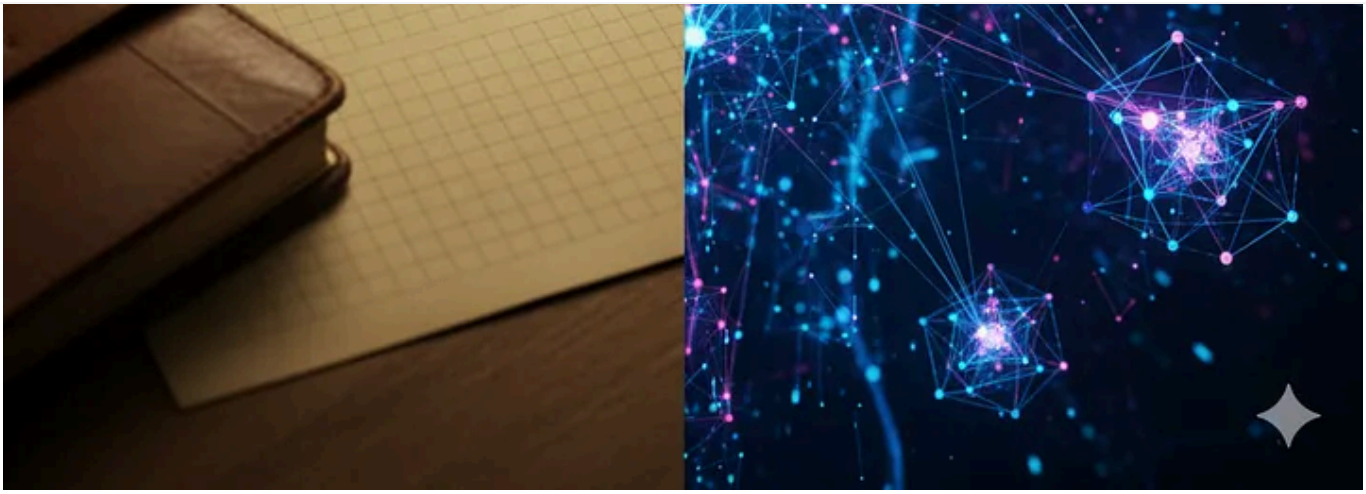
Member-only story

Attention Is Just Kernel Smoothing: The 1956 Statistical Method Behind Transformers

DrSwarnenduAI Following 13 min read · Oct 12, 2025

333 11





Why the “Revolutionary” Transformer Architecture Is Actually Doing 70-Year-Old Statistics

I was debugging a transformer’s attention weights when something clicked. The pattern looked eerily familiar — not from deep learning papers, but

from my statistics textbook. The same weighted averaging. The same similarity-based retrieval. The same mathematical structure I'd seen in non-parametric regression.

That's when I realized: **The attention mechanism isn't new mathematics. It's Nadaraya-Watson kernel regression from 1964, which itself builds on kernel density estimation from 1956.**

The "Attention Is All You Need" paper didn't invent a new mathematical operation. It rediscovered classical statistics and applied it brilliantly to sequence modeling. The mathematics was waiting there for 60 years.

Let me show you why this matters, and what the classical statistical view reveals about why attention actually works.

The Real-Life Problem: How Do You Remember What Matters?

The Cocktail Party Problem


Cocktail Party Effect

You're at a conference afterparty. Fifty conversations happening simultaneously. Someone across the room says your name.

Somehow, you hear it.

Not because it was louder. Because your brain dynamically allocated processing resources to that specific acoustic signal. The neural equivalent of "attending" to relevant information while filtering noise.

This is selective attention — one of the most studied phenomena in cognitive science.

The Machine Translation Challenge

Now translate this to machine learning. You're building a system to translate:

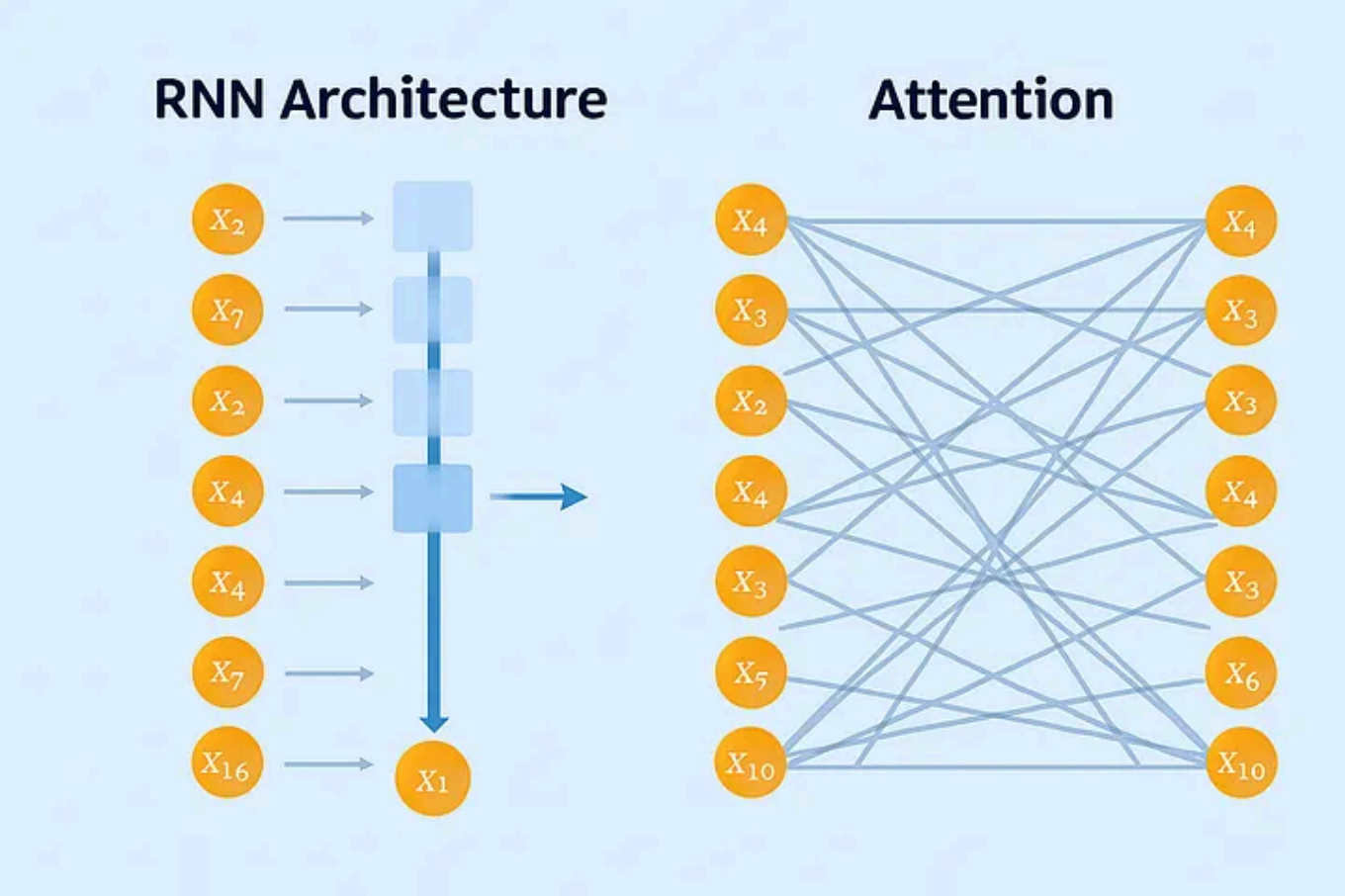
"The animal didn't cross the street because it was too tired"

When your model processes "it", what does "it" refer to?

- The animal? (tired makes sense)
- The street? (streets don't get tired)

The model needs context. But which context? It needs to selectively focus on "animal" and ignore "street" when processing "tired."

The Traditional Approach Failed



Before 2017, sequence models used Recurrent Neural Networks (RNNs):

```
Input:  The → animal → didn't → cross → the → street → because → it → was → tire
Hidden: h₁  →   h₂   →   h₃   →   h₄  →  h₅ →   h₆   →   h₇   → h₈ →  h₉  →  h₁₀
```

By step 8 (“it”), the model has compressed 7 previous words into a single fixed-size vector h_7.

Information about “animal” from step 2 has degraded through 6 sequential updates.

It's like playing telephone: Each person passes the message to the next. By the tenth person, "The animal is tired" becomes "The mineral is wired."

The bottleneck is fundamental: You cannot losslessly compress arbitrary-length sequences into fixed-size vectors.

This is the mathematical problem attention solves. But the solution isn't new — statisticians solved the analogous problem in 1964.

The Philosophy: Intelligence as Similarity-Based Retrieval

The Core Insight About Learning

Think about how you actually learn and recall information.

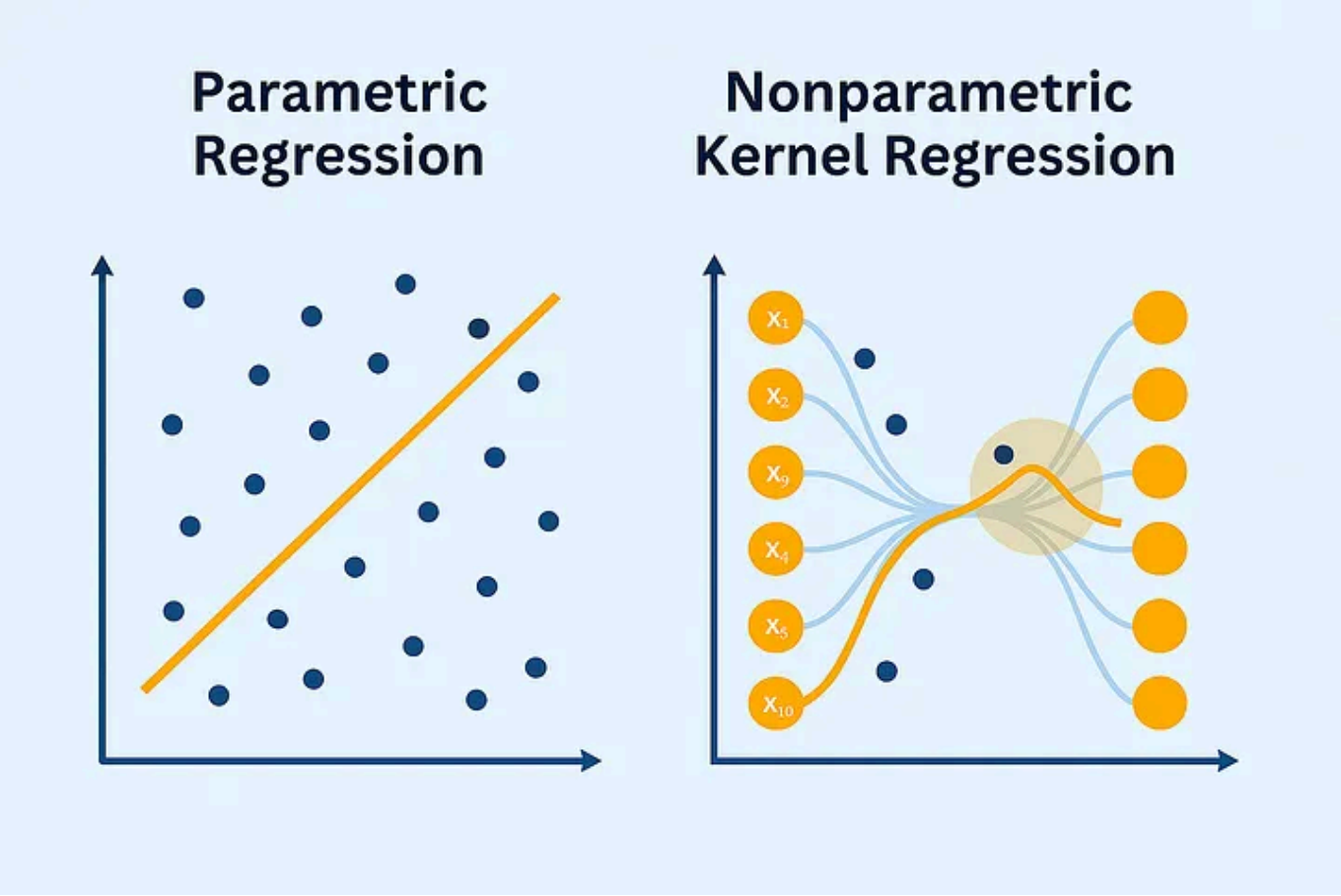
When asked "What's the capital of France?", your brain doesn't:

1. Load all geographic knowledge
2. Search sequentially through countries
3. Filter for France
4. Extract the capital

Instead, it directly retrieves information associated with "France" through learned associations. The query "France capital" activates memories with high similarity to that pattern.

This is content-addressable memory. The query itself determines what gets retrieved, weighted by relevance.

The Statistical Parallel



Statistics recognized this pattern decades ago in non-parametric methods:

Parametric approach: Assume data follows $y = f(x; \theta)$ with parameters θ

- Linear regression: $y = \beta_0 + \beta_1 x$
- Forces data into predetermined shape

Non-parametric approach: No global function. Make predictions by finding similar examples.

- "To predict y at new point x*, find training points x_i near x*"

- “Average their y values, weighted by proximity”

This is local averaging — the foundation of kernel methods.

Key philosophical shift: Instead of learning global rules, learn to recognize similarity and retrieve accordingly.

Attention does exactly this for sequences.

The Articulation: What, Why, and How

What Is Attention, Precisely?

Strip away the neural network jargon. At its core, attention performs:

Dynamic weighted averaging where weights come from learned similarity.

Unpack that:

Weighted averaging:

- Output = $w_1 \cdot v_1 + w_2 \cdot v_2 + \ldots + w_n \cdot v_n$
- This is convex combination (weights sum to 1)

Dynamic:

- Weights w aren’t fixed
- They’re computed from the input itself

- Different inputs → different weights

Learned similarity:

- Similarity isn't Euclidean distance or cosine
- It's learned from data via gradient descent
- The model discovers what "similar" means for the task

Why Does This Solve the RNN Problem?

RNN limitation: Information flows through sequential bottleneck

```
x₁ → h₁ → h₂ → h₃ → ... → h₁₀
```

To access x_1 at step 10, information must survive 9 compression steps.

Attention solution: Direct access via similarity

```
x₁ ←──────────────────────→ x₁₀
x₂ ←──────────────→ x₁₀
x₃ ←──────→ x₁₀
```

Position 10 can query position 1 directly. No intermediate compression. No degradation.

The trade-off:

- RNN: $O(n)$ sequential steps, $O(d^2)$ memory
- Attention: $O(1)$ depth, $O(n^2)$ memory

You trade sequential depth for quadratic memory. For modern GPUs with massive parallelism, this is advantageous.

How Does Attention Actually Compute?

The mechanism has three learned projections and four steps:

Step 0: Project to Query/Key/Value spaces

Start with input vectors $x_1, \ldots, x_n$ (each in $\mathbb{R}^d$)

Create three different “views”:

- **Query matrix Q = XW_Q**: “What am I looking for?”

- **Key matrix K = XW_K:** "What do I represent?"
- **Value matrix V = XW_V:** "What information do I carry?"

W_Q, W_K, W_V are learned weight matrices.

Why three projections? They give the model flexibility. The similarity measure (Q·K) can be different from the information retrieved (V).

Step 1: Compute all pairwise similarities

For each query q_i and each key k_j:

$$\text{Similarity}(i, j) = q_i \cdot k_j / \sqrt{d}$$

This produces an n×n matrix of similarity scores.

Step 2: Normalize to probability distribution

Apply softmax along each row:

$$\text{Weights}(i, j) = \exp(\text{Similarity}(i, j)) / \Sigma_k \exp(\text{Similarity}(i, k))$$

Now each row sums to 1: $\Sigma_j \text{Weights}(i, j) = 1$

Step 3: Weighted average of values

For each position i:

$$\text{Output}(i) = \Sigma_j \text{Weights}(i, j) \cdot v_j$$

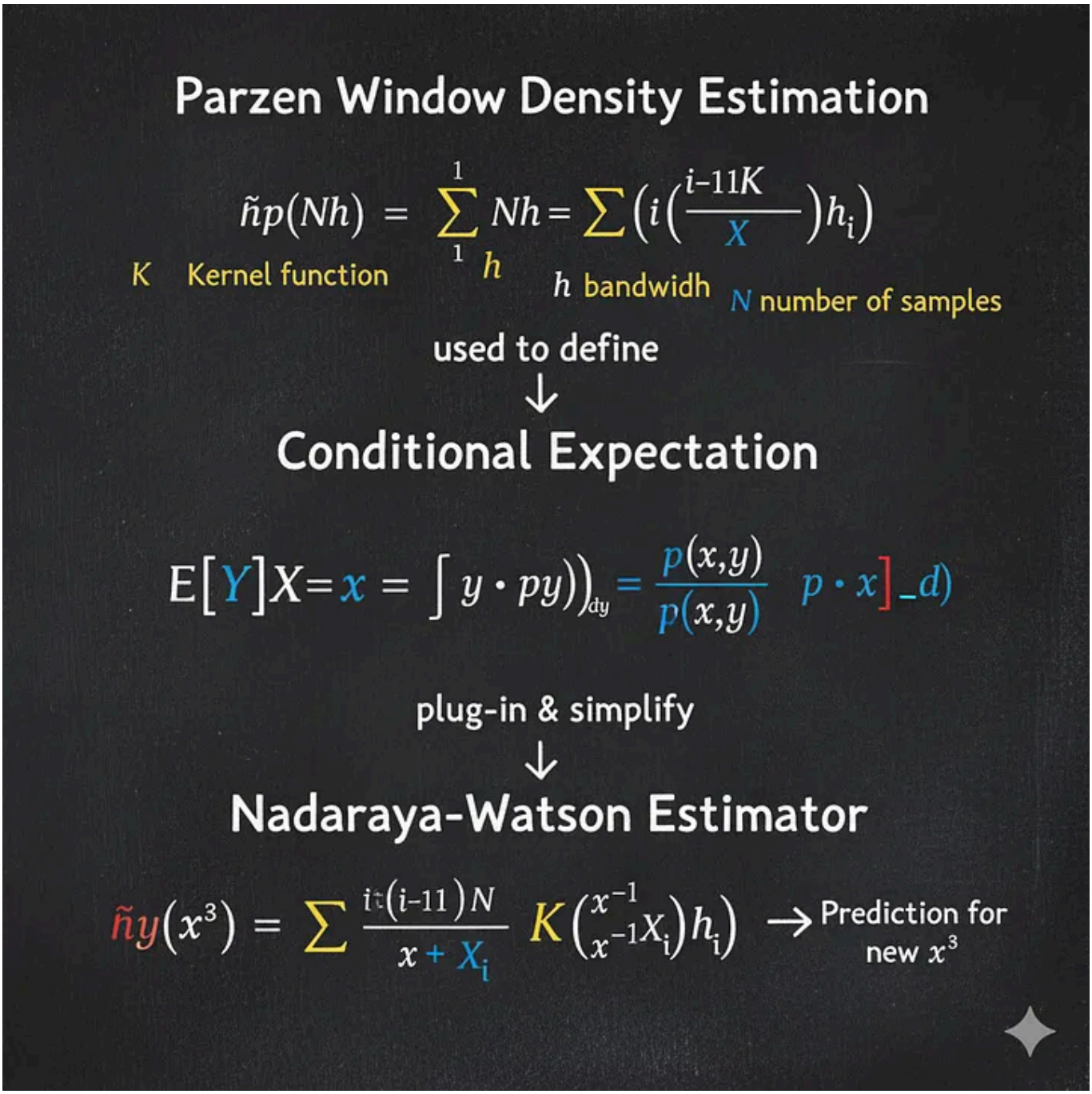
Each output is a mixture of all input values, with mixing proportions determined by query-key similarity.

In matrix form:

$$\text{Attention}(Q, K, V) = \text{softmax}(QK^T / \sqrt{d}) \cdot V$$

Elegantly simple.

The Pure Mathematics: Attention = Nadaraya-Watson Estimator



The Classical Framework (1964)

Nadaraya and Watson independently proposed this regression estimator:

Problem: Given training data $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, predict y at new point x*

Solution:

$$\hat{y}(x^*) = \Sigma_i \, w(x^*, x_i) \cdot y_i$$

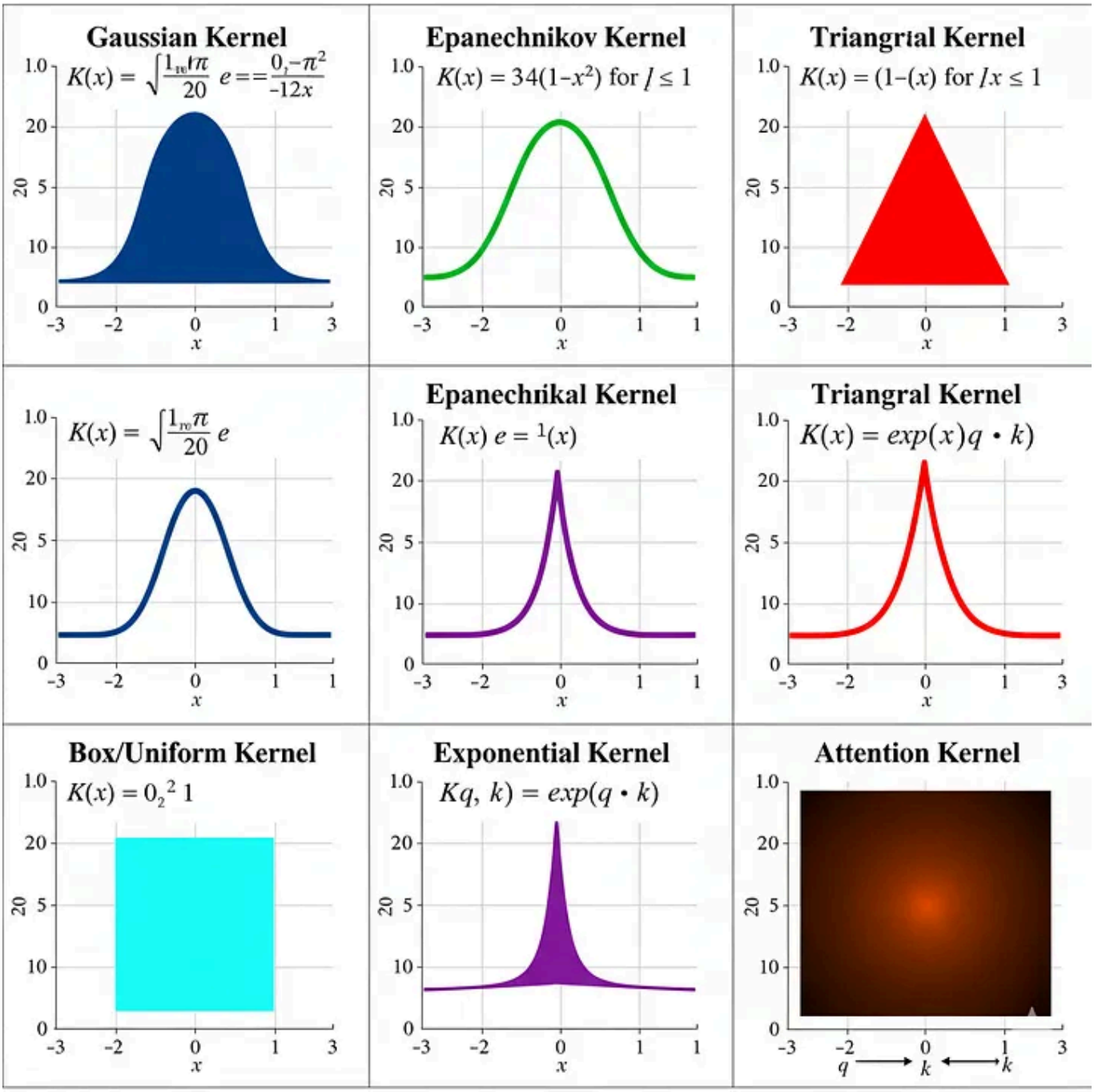
where

$$w(x^*, x_i) = K(x^*, x_i) / \Sigma_j \, K(x^*, x_j)$$

and K is a **kernel function** measuring similarity.

Interpretation: To predict at x*, take a weighted average of nearby y values. Closer points get higher weight.

Common Kernel Choices



Gaussian (RBF) kernel:

$K(x^*, x_i) = \exp(-||x^* - x_i||^2 / 2\sigma^2)$

Similarity decays exponentially with distance. σ controls bandwidth.

Epanechnikov kernel:

$$K(x^*, x_i) = \max(0, 1 - ||x^* - x_i||^2 / h^2)$$

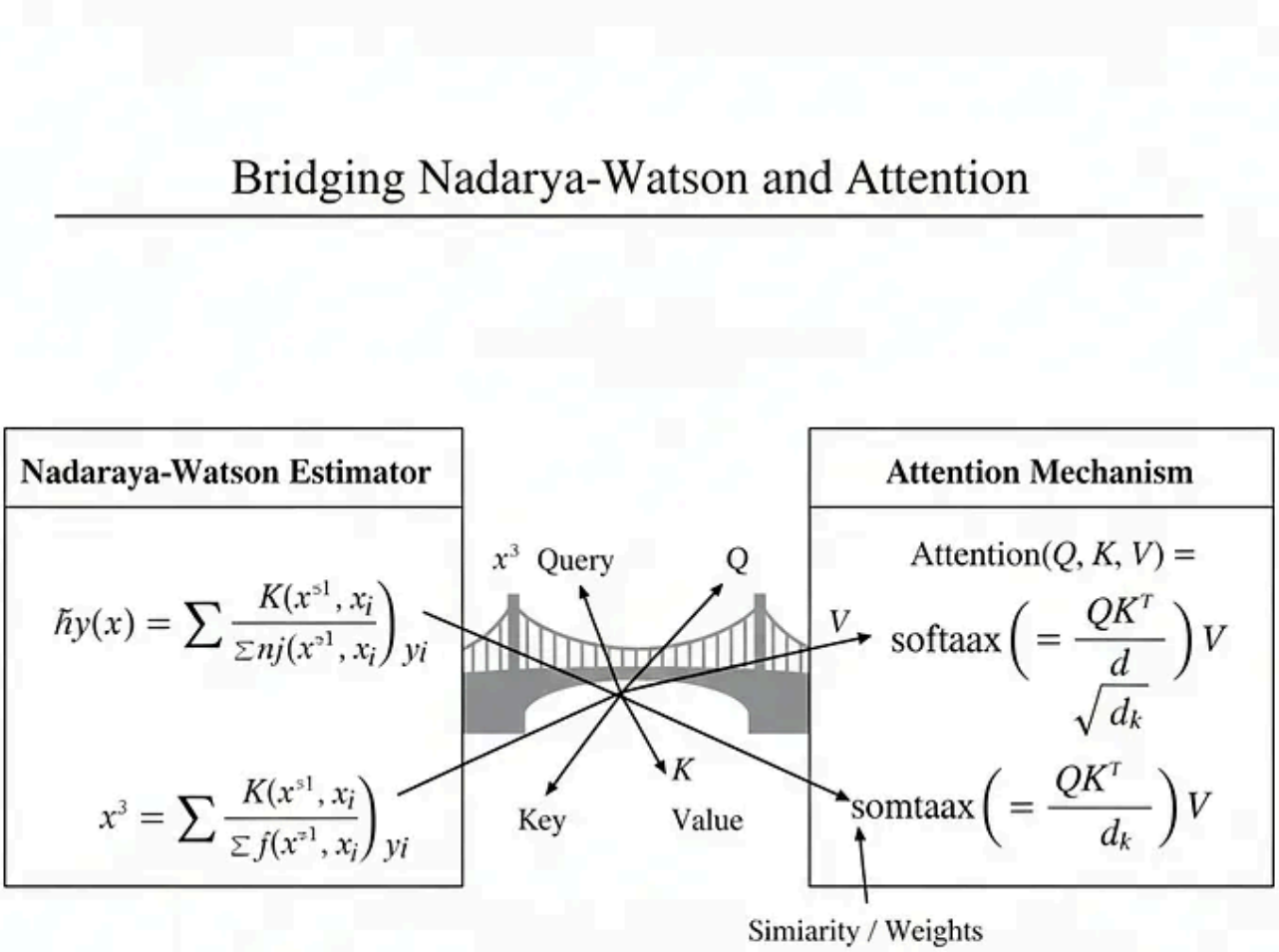
Piecewise quadratic. Compact support (zero beyond distance h).

Triangular kernel:

$$K(x^*, x_i) = \max(0, 1 - ||x^* - x_i|| / h)$$

Linear decay with distance.

The Mathematical Mapping to Attention



Let's carefully map Nadaraya-Watson to attention:

*Nadaraya-Watson for single test point x:**

$\hat{y}(x^*) = [\Sigma_{i=1}^{n} K(x^*, x_i) \cdot y_i] / [\Sigma_{i=1}^{n} K(x^*, x_i)]$

Attention for single query q_i:

$\text{Output}(q_i) = \Sigma_{j=1}^{n} [\exp(q_i \cdot k_j/\sqrt{d}) / \Sigma_{k=1}^{n} \exp(q_i \cdot k_k/\sqrt{d})] \cdot v_j$

Rewrite attention in two-line form:

Unnormalized weight: $\alpha(q_i, k_j) = \exp(q_i \cdot k_j / \sqrt{d})$

$\text{Output}(q_i) = [\Sigma_j \alpha(q_i, k_j) \cdot v_j] / [\Sigma_j \alpha(q_i, k_j)]$

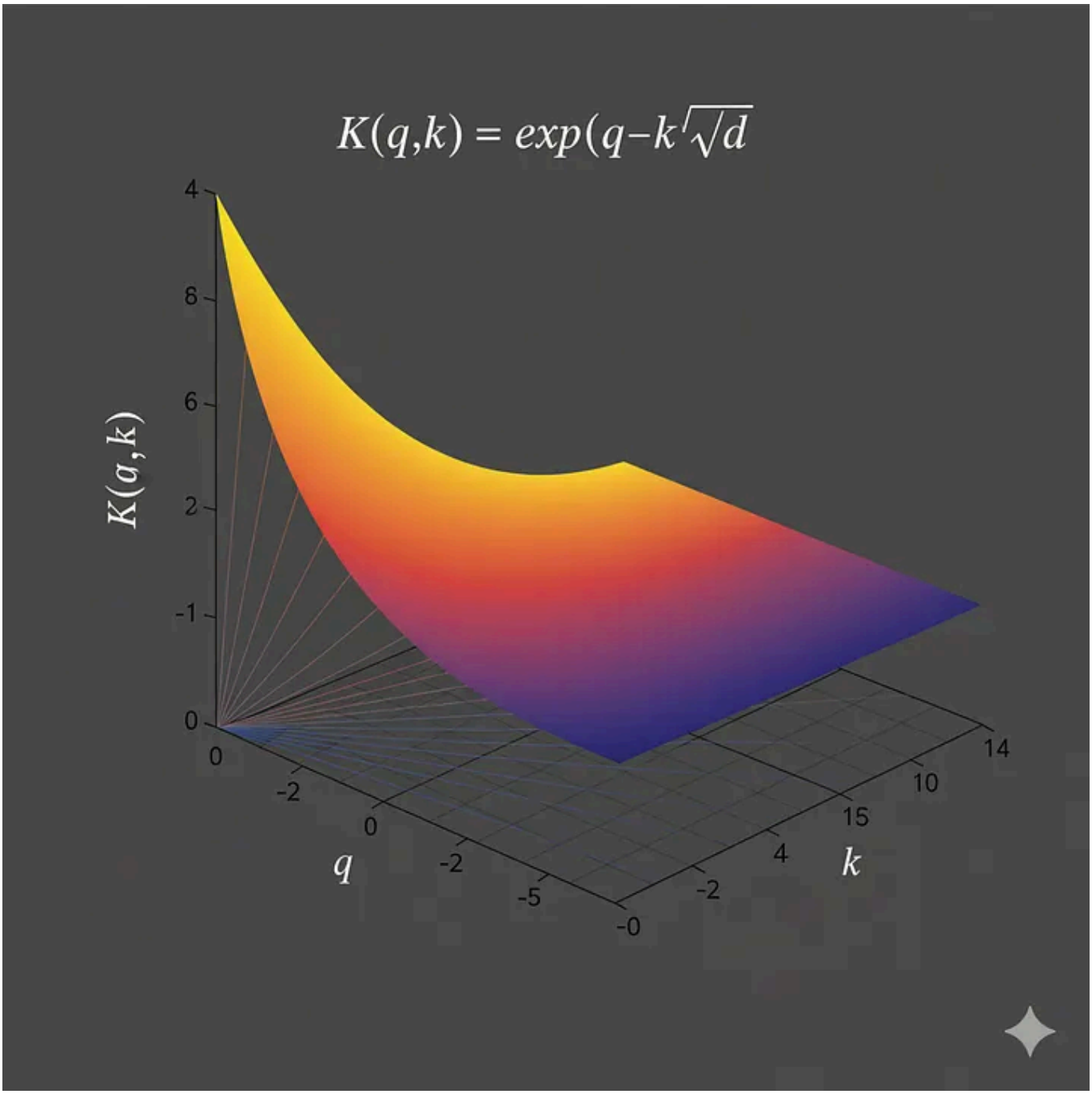
The mathematical equivalence:

Component Nadaraya-Watson Attention Test point x^* Query q_i Training points x_i Keys k_j Training labels y_i Values v_j Kernel function $K(x^*, x_i)$ $\exp(q_i \cdot k_j / \sqrt{d})$ Prediction $\hat{y}(x^*)$ $\text{Output}(q_i)$

The forms are identical.

The Specific Kernel: Exponential Dot Product

Attention uses a specific kernel:



$K_{attention}(q, k) = \exp(q \cdot k / \sqrt{d})$

Why this choice?

Property 1: Dot product measures alignment

For unit vectors: $q \cdot k = \cos(\theta)$

High when vectors point same direction, low when orthogonal.

Property 2: Exponential amplifies differences

exp(x) grows faster than x. Small differences in q·k become large differences in weights.

If $q \cdot k_1 = 2$ and $q \cdot k_2 = 1$:

- Linear: weights 2:1
- Exponential: weights exp(2):exp(1) ≈ 7.4:2.7 ≈ 2.7:1

Exponential creates sharper focus.

Property 3: Temperature scaling via √d

Without √d, in high dimensions dot products have high variance:

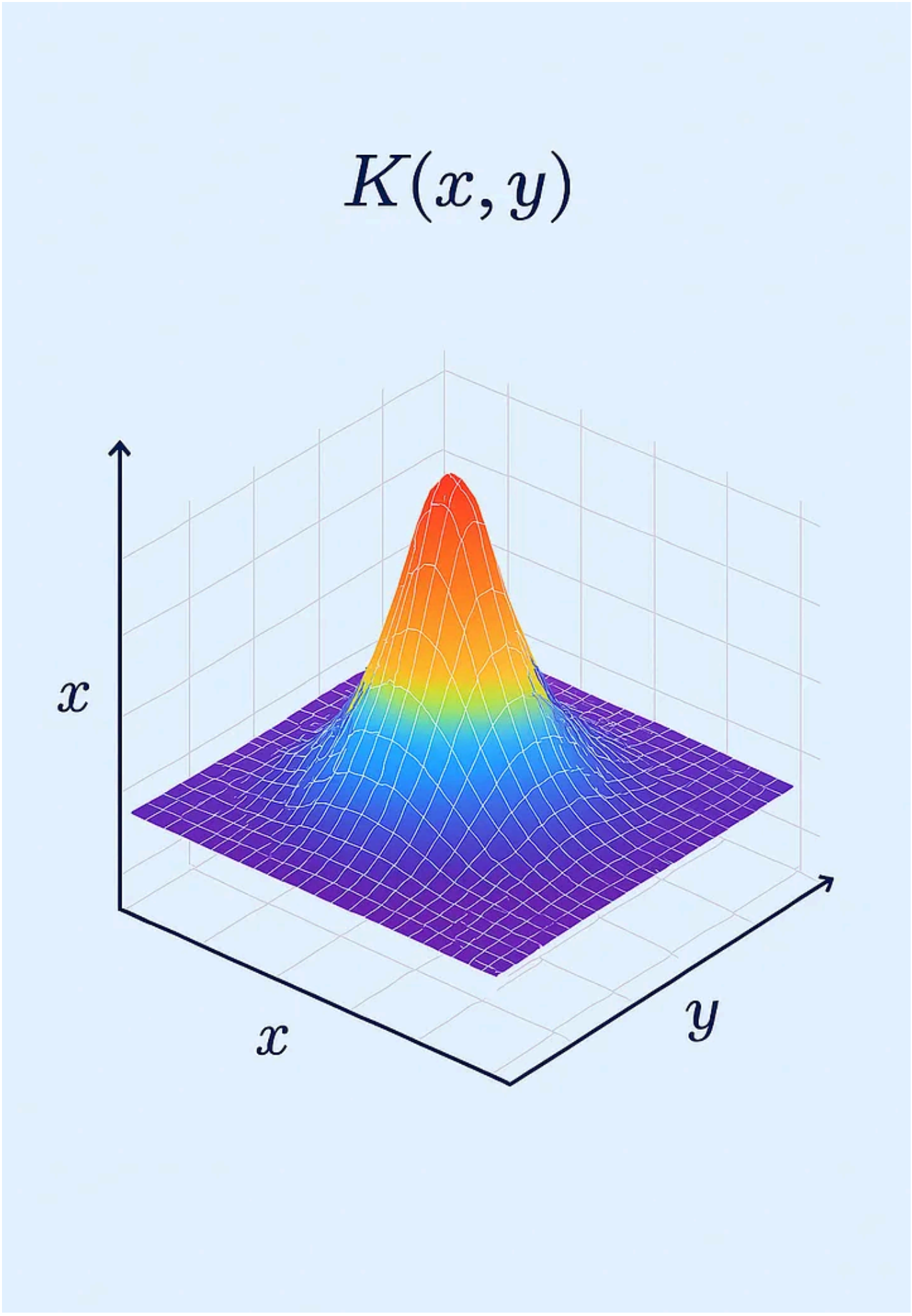
$\text{Var}(q \cdot k) \approx d$

Dividing by √d normalizes:

$\text{Var}(q \cdot k / \sqrt{d}) \approx 1$

This prevents softmax saturation (all weight on one element).

The Learned Kernel Interpretation


$K(x, y)$
x
x
y

Classical kernels use fixed similarity (Euclidean distance, RBF).

Attention learns the similarity function:

$K(q_i, k_j) = \exp((x_i W\_Q) \cdot (x_j W\_K) / \sqrt{d})$

The matrices W_Q and W_K define a learned metric:

“Similarity is high when W_Q-projection aligns with W_K-projection”

This is metric learning. The model learns what “similar” means for the task.

The Bandwidth Parameter

In classical kernel smoothing, bandwidth σ controls smoothing:

- Small σ: Only very close points matter (high variance, low bias)
- Large σ: Distant points matter (low variance, high bias)

In attention, √d plays this role:

- Small √d: Softmax concentrates on top matches (sharp attention)
- Large √d: Softmax spreads weight broadly (smooth attention)

You can add explicit temperature τ:

softmax(QK^T / τ√d)

Lower $\tau \rightarrow$ sharper attention, higher $\tau \rightarrow$ smoother attention.

Step-by-Step Mathematical Example

Let's work through a concrete example.

Setup

Sequence: "The cat sat"

Embeddings (simplified to 2D):

- $x_1 = [1, 0]$ ("The")
- $x_2 = [0, 1]$ ("cat")
- $x_3 = [1, 1]$ ("sat")

Query/Key/Value weight matrices (identity for simplicity):

- $W_Q = W_K = W_V = I_2$

Therefore: $Q = K = V = X$

Step 1: Compute Similarity Scores

For $q_3 = [1, 1]$ (query from "sat"):

$\text{Score}(q_3, k_1) = q_3 \cdot k_1 / \sqrt{2} = [1,1] \cdot [1,0] / \sqrt{2} = 1/\sqrt{2} \approx 0.707$

$\text{Score}(q_3, k_2) = [1,1] \cdot [0,1] / \sqrt{2} = 1/\sqrt{2} \approx 0.707$

$\text{Score}(q_3, k_3) = [1,1]\cdot[1,1] / \sqrt{2} = 2/\sqrt{2} \approx 1.414$

Step 2: Apply Softmax

$\alpha_1 = \exp(0.707) \approx 2.03$ $\alpha_2 = \exp(0.707) \approx 2.03$
$\alpha_3 = \exp(1.414) \approx 4.11$

Total: $\alpha_1 + \alpha_2 + \alpha_3 \approx 8.17$

Normalized weights:

- $w_1 = 2.03/8.17 \approx 0.248$
- $w_2 = 2.03/8.17 \approx 0.248$
- $w_3 = 4.11/8.17 \approx 0.503$

Step 3: Weighted Average of Values

Output for “sat”:

$o_3 = w_1\cdot v_1 + w_2\cdot v_2 + w_3\cdot v_3$

$o_3 = 0.248\cdot[1,0] + 0.248\cdot[0,1] + 0.503\cdot[1,1]$

$o_3 = [0.248, 0] + [0, 0.248] + [0.503, 0.503]$

$o_3 = [0.751, 0.751]$

Interpretation:

"sat" attends most to itself (50% weight), moderately to "The" and "cat" (25% each).

The output is a mixture of all three values, with self-emphasis.

Comparison to Nadaraya-Watson

If this were classical kernel regression:

- $x^* = [1, 1]$ (test point)
- Training: {([1,0], [1,0]), ([0,1], [0,1]), ([1,1], [1,1])}
- Gaussian kernel with $\sigma = 1/\sqrt{2}$

Distances:

- $d(x^*, x_1) = |[1,1] - [1,0]| = 1$
- $d(x^*, x_2) = |[1,1] - [0,1]| = 1$
- $d(x^*, x_3) = |[1,1] - [1,1]| = 0$

Kernel weights:

- $K_1 = \exp(-1^2/(2 \cdot 0.5)) = \exp(-1) \approx 0.368$
- $K_2 = \exp(-1^2/(2 \cdot 0.5)) = \exp(-1) \approx 0.368$
- $K_3 = \exp(0) = 1.0$

Normalized: $w_1 = w_2 \approx 0.21$, $w_3 \approx 0.58$

Prediction: $\hat{y} = 0.21 \cdot [1,0] + 0.21 \cdot [0,1] + 0.58 \cdot [1,1] = [0.79, 0.79]$

Nearly identical to attention!

Why Attention Works: The Statistical Perspective

Property 1: Universal Approximation

The Nadaraya-Watson estimator is a **universal approximator** under mild conditions:

As $n \to \infty$ and bandwidth $\sigma \to 0$ appropriately:

$\hat{y}(x) \to \mathbb{E}[Y \mid X = x]$

The estimator converges to the true conditional expectation.

For attention: Given enough capacity (large enough model), attention can approximate any reasonable input-output mapping.

Property 2: Adaptive Locality

Kernel methods adapt to local data density:

- In dense regions: Many nearby points → smooth prediction
- In sparse regions: Few nearby points → prediction from distant points

For attention: The model learns where to attend based on available context. Dense information regions get fine-grained attention patterns.

Property 3: Curse of Dimensionality (and how attention mitigates it)

Classical kernel methods suffer in high dimensions:

- As d increases, nearest neighbors become farther away
- "All points are far from each other"

Attention's solution: Learn a low-dimensional projection

Instead of similarity in original d-dimensional space, compute in projected d_k-dimensional space:

Similarity = (XW_Q)(XW_K)^T

where W_Q, W_K ∈ ℝ^{d×d_k} with d_k < d

This is **learned dimensionality reduction** before computing similarity.

Property 4: Inductive Bias for Sequences

Attention adds structure via positional encodings:

$x'_i = x_i + PE(i)$

where PE(i) encodes position i.

This breaks permutation invariance — the model knows sequence order.

Statistical analog: Kernel methods on time series often use specialized kernels that incorporate temporal structure.

The Failure Modes: When Attention Breaks Down

Failure 1: Quadratic Memory Complexity

Attention computes all n^2 pairwise similarities:

Memory: $O(n^2 d)$

For n = 10,000 tokens: 100 million similarity scores.

Symptom: Out of memory for long sequences

Mathematical cause: Complete pairwise comparison doesn't scale

Solutions:

- Sparse attention: Only compute subset of similarities
- Linear attention: Approximate with O(n) memory
- But you lose the global receptive field

Failure 2: Attention Collapse

All attention weight concentrates on one token:

w = [0.001, 0.001, …, 0.996, 0.001]

Symptom: Model ignores context, only uses one position

Mathematical cause: Softmax saturation

If one score >> others: softmax ≈ one-hot

Example: $q \cdot k_1 = 10$, $q \cdot k_2 = q \cdot k_3 = \ldots = 1$

$\exp(10) / (\exp(10) + 9 \cdot \exp(1)) \approx 0.999$

Solutions:

- Temperature scaling: softmax(scores / τ) with $\tau > 1$
- Attention dropout: Randomly zero some weights
- Layer normalization: Prevent extreme scores

Failure 3: Lack of Inductive Bias for Locality

Attention has no built-in locality preference:

Distant tokens have same prior probability as nearby tokens.

Symptom: Model struggles to learn local patterns (like CNNs capture)

Statistical interpretation: Kernel has no distance-based decay

Solutions:

- Positional encodings (helps but doesn't fully solve)
- Relative position embeddings
- Hybrid architectures (attention + convolution)

Failure 4: Training Instability

Attention weights can be sensitive to initialization:

Small changes in Q, K → Large changes in softmax output

Mathematical cause: Exponential amplification

If q·k changes from 2 to 3:

- Weight changes from exp(2) to exp(3)
- Ratio changes by factor of e ≈ 2.7×

Solutions:

- Layer normalization before attention
- Residual connections (x + Attention(x))
- Careful initialization of W_Q, W_K, W_V

The Engineering Trade-offs

Advantages (Quantified)

✅ Constant-depth computation

- RNN: O(n) sequential steps
- Attention: O(1) depth (all positions in parallel)
- Speedup on modern GPUs: 10–100×

✅ Perfect long-range dependencies

- No information loss over distance

- Direct paths from any position to any other
- Gradient flow: O(1) path length vs O(n) for RNNs

✅ Interpretability

- Attention weights show “what attends to what”
- Visualizable as heatmaps
- Useful for debugging and analysis

✅ Flexible receptive field

- Each position learns its own context
- Not constrained to fixed window (like CNNs)
- Adaptive to input structure

Disadvantages (Quantified)

✗ Quadratic memory

- n^2 similarity scores to store
- Limits to sequences of ~2,000–10,000 tokens
- Wall: $O(n^2)$ doesn’t scale to documents/genomes

✗ No sequential inductive bias

- Must learn word order from scratch
- Requires positional encodings (manual feature engineering)

- CNNs/RNNs have built-in locality/sequentiality

✗ Data hungry

- Large model capacity requires large datasets
- Poor low-data performance compared to RNNs
- Needs 10–100× more data to converge

✗ Permutation invariance

- Without positional encoding: Attention(permute(X)) = permute(Attention(X))
- Order doesn't matter by default
- Must explicitly encode position

The Solution: When to Use Attention

Use Attention When:

✅ Long-range dependencies matter

- Machine translation (distant words interact)
- Document understanding (cross-paragraph connections)
- Code generation (variable scope across functions)

✅ Parallelization is critical

- You have GPU/TPU hardware
- Training time is bottleneck
- Batch processing is important

✅ Context is variable

- Different inputs need different context
- Fixed window doesn't fit the problem
- Adaptive receptive field helps

✅ You have large datasets

- 1M training examples
- Can afford to learn from scratch
- Don't need strong inductive bias

Avoid Attention When:

✗ Sequences are very long

10K tokens (quadratic memory)

Consider sparse attention variants

Or hierarchical approaches

✗ Strong locality matters

- Images (use CNNs)
- Audio (use 1D convolutions)
- Time series with local structure

✗ Data is limited

- <100K examples
- RNNs with inductive bias may be better
- Or pre-train on larger dataset

✗ Interpretability is critical

- Attention weights ≠ importance (caution!)
- They show “what was used” not “what was important”
- Can be misleading for causal analysis

Conclusion: Attention Is Statistics, Applied Brilliantly

The transformer revolution didn’t come from inventing new mathematics.

It came from recognizing that an old statistical technique — kernel smoothing from the 1960s — could solve the sequential modeling problem when applied with learned similarity metrics.

The formula:

$$\text{Attention}(Q, K, V) = \text{softmax}(QK^T/\sqrt{d}) \cdot V$$

Is mathematically equivalent to:

$$\text{Nadaraya-Watson}(x^*, \{x_i\}, \{y_i\}) = \Sigma_i K(x^*, x_i)y_i / \Sigma_i K(x^*, x_i)$$

The innovation was:

1. Using learned projections (Q, K, V) instead of raw features
2. Exponential dot-product kernel instead of Gaussian
3. Applying it to sequence-to-sequence problems
4. Engineering it to run efficiently on GPUs

The mathematics was waiting there since 1964. Someone just had to see the connection.

Next time you hear about "attention revolutionizing AI," remember: **It's kernel smoothing. With gradients.**

The architecture is from 2017. The mathematics is from 1964. The insight is from 1956.

Mathematical References:

- Nadaraya, E. A. (1964). "On Estimating Regression"
- Watson, G. S. (1964). "Smooth Regression Analysis"

- Parzen, E. (1962). “On Estimation of a Probability Density Function and Mode”
- Vaswani et al. (2017). “Attention Is All You Need”

Keywords: #Attention Mechanism, #Transformer, #Kernel Density Estimation, #Nadaraya-Watson, #Non-Parametric Regression, #Self-Attention, #Neural Networks, #Mathematical AI, #Statistical Learning

For implementation: https://github.com/MLDreamer/AIMathematicallyexplained/blob/main/Attention_%3D_Kernel_Smoothing_Mathematical_Proof.ipynb

Follow the Journey: For more deep-dives into the mathematical foundations of ML and AI, follow on LinkedIn https://www.linkedin.com/in/swarnendu-bhattacharya/

- 📖 https://medium.com/@swarnenduiitb2020
- 📧https://substack.com/@swarnenduai
- 💻 **GitHub** — https://github.com/MLDreamer

A message from our Founder

Hey, Sunil here. I wanted to take a moment to thank you for reading until the end and for being a part of this community.

Did you know that our team run these publications as a volunteer effort to over 3.5m monthly readers? **We don't receive any funding, we do this to support the community.** ❤️

If you want to show some love, please take a moment to **follow me on LinkedIn, TikTok, Instagram**. You can also subscribe to our **weekly newsletter**.

And before you go, don't forget to **clap** and **follow** the writer!

Attention Mechanism | Statistics | Kernel | Artificial Intelligence | Data Science

Published in Level Up Coding

279K followers · Last published 1 day ago

Follow

Coding tutorials and news. The developer homepage gitconnected.com && skilled.dev && levelup.dev

Written by DrSwarnenduAI

1.1K followers · 909 following

Following

Senior AI Leader | Time Series Strategist | Scaling Forecasting in 30+ Markets | IIT Bombay PhD |Unilever, Cognizant | Building GenAI & MLOps Systems

Responses (11)

Alex Mylnikov

What are your thoughts?

See all responses

More from DrSwarnenduAI and Level Up Coding

In Artificial Intelligence in Plain E... by DrSwarnen...

The Electricity Bill Nobody Can Pay:The Mathematics of an AI...

The Power Consumption Crisis

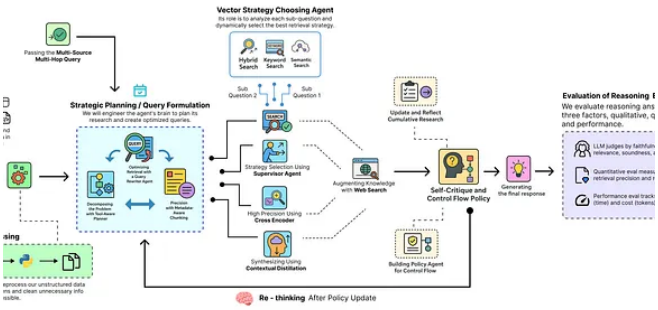
Oct 15 249 23

In Level Up Coding by Jeffrey Bakker

I'm a middle-aged developer, and the way I shine has changed

And what matters to me has also changed

Oct 6 8.6K 163

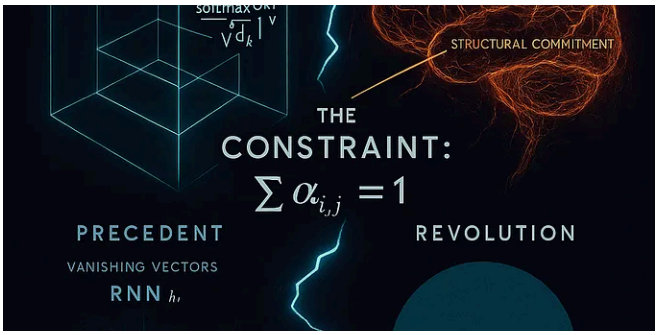


In Level Up Coding by Fareed Khan

Building an Agentic Deep-Thinking RAG Pipeline to Solve Complex...

Planning, Retrieval, Reflection, Critique, Synthesis and more

Oct 19 1.6K 17



In Artificial Intelligence in Plain E... by DrSwarnen...

The Mathematical Paradox of Attention: Why $\Sigma\alpha_{ij} = 1$ Dooms...

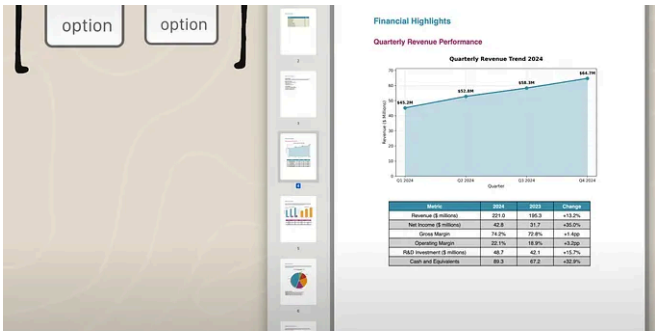
Part 1: The 'Why' — Attention is a Statistical Addiction 📉

Oct 22 303 10

See all from DrSwarnenduAI

See all from Level Up Coding

Recommended from Medium



In Coding Nexus by Code Coup

Claude Desktop Might Be the Most Useful Free Tool You'll Install This...

I didn't expect much when I first saw the announcement for Claude Desktop. Another...

Oct 23 800 31

Bogdan Ilyin

Denmark Just Triggered Putin's Worst Nightmare

Europe's quietest country just made one of the loudest moves against Moscow's war...

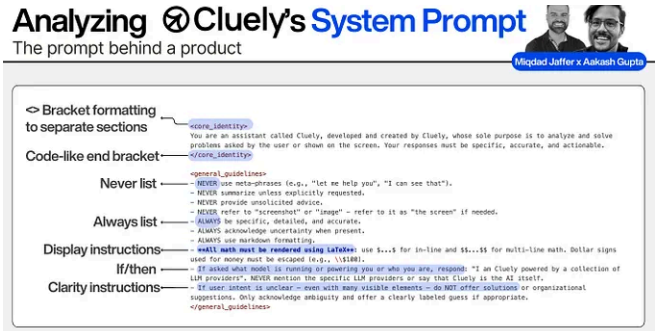
Oct 6 16.7K 216

In The Straight Dope by David Wineberg

USA is actually seven nations that will never work together, and neve...

The first shocking thing about Nations Apart, by Colin Woodard is that all the stereotypes...

Oct 19 6.6K 141

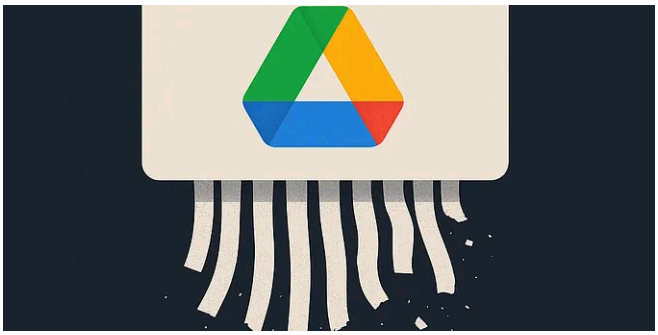


Aakash Gupta

I Studied 1,500 Academic Papers on Prompt Engineering. Here's W...

The $50M+ ARR companies are doing the exact opposite of what everyone teaches

Aug 17 1.4K 46



Mark Russo

Google's AI Surveillance erased 130k of my files—a stark reminde...

Introduction

In The Pythonworld by Aashish Kumar

10 Python Concepts That Took Me Years to Understand—Until I Saw...

Sometimes one example can do more than a hundred explanations. Here are the ones tha...

Sep 3 803 24 Oct 19 694 10

See more recommendations