## Transformers: Understanding Parallel Intelligence
From Zero to ChatGPT - A BSc Journey

Week 5: Transformers

# What You'll Learn Today

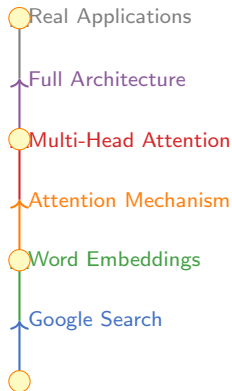**By the end of this lecture, you will:**

1. **Understand** how words become numbers (embeddings)
2. **Explain** why parallel beats sequential processing
3. **Calculate** attention scores between words
4. **Draw** the transformer architecture
5. **Identify** transformers in daily applications

**Prerequisites:**

- Basic vector operations (dot product)
- Matrix multiplication concept
- No deep learning needed!

**Your Learning Journey:**

Real Applications

Full Architecture

Multi-Head Attention

Attention Mechanism

Word Embeddings

Google Search

**Interactive Elements:**
3 Checkpoints — 5 Exercises — 10 Visuals

**Try this:** Type in Google: "How do transformers..."

**Google instantly suggests:**
- "...work in machine learning"
- "...process language"
- "...learn from data"

**The Mystery:**
- Google reads ALL your words at once
- Not word-by-word like old systems
- Understands context instantly

```
How do transformers

_____

...work in machine learning
...process language
...learn from data
...handle attention
```

**Question:** How does it understand whole sentences simultaneously?
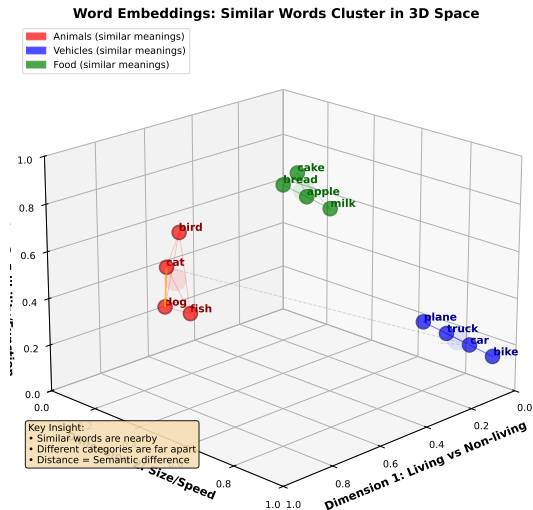
# Discovery 1: Words Live in Space

**Think about GPS coordinates:**

- Paris: (48.8N, 2.3E, 35m altitude)
- London: (51.5N, 0.1W, 11m altitude)
- Similar cities are nearby in space

**Words work the same way!**

- "cat": [0.7, 0.2, 0.5] in meaning space
- "dog": [0.8, 0.3, 0.4] (nearby - similar!)
- "car": [0.1, 0.9, 0.2] (far - different!)

**This is called:** Word Embeddings



Word Embeddings: Similar Words Cluster in 3D Space

# Discovery 2: Every Word Connects to Every Other

**In a sentence, every word "talks" to every other:**

**Example:** "The cat sat on mat"

- "cat" checks all other words
- "sat" looks at subject and location
- "mat" knows what's on it

**Total connections:** $n \times n$

- 5 words = 25 connections
- 100 words = 10,000 connections!



Every Word Connects to Every Other: 5×5 = 25 Connections

Total Connections: 25
Self-connections: 5
Cross-connections: 20

PARALLEL PROCESSING

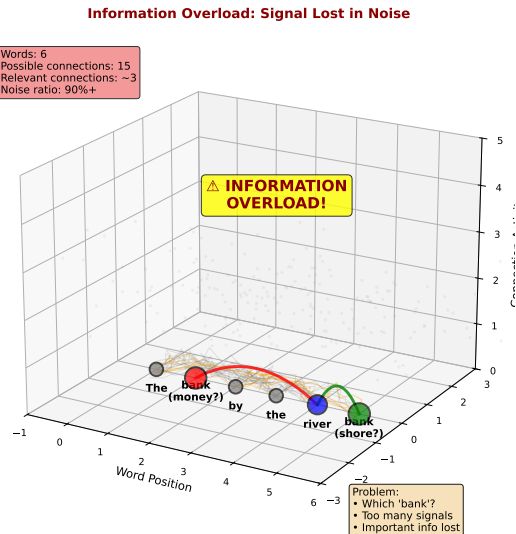Key: All connections computed SIMULTANEOUSLY in transformers!
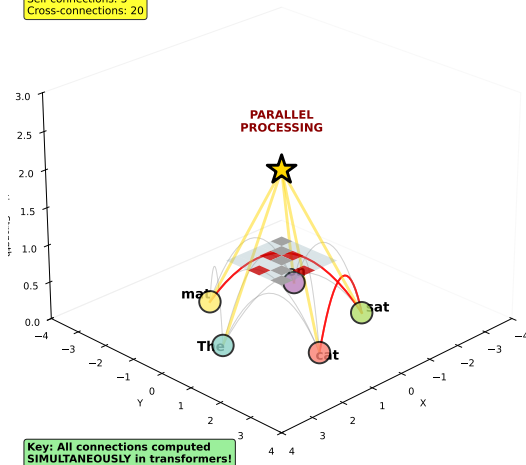
**Challenge:** Too much information!

**Sentence:** "The bank by the river bank"

- First "bank" = financial institution
- Second "bank" = river edge
- How does the model know?

**Information explosion:**

- Every word sends signals to all others
- Most connections are noise
- Need to focus on what matters



Information Overload: Signal Lost in Noise

Words: 6
Possible connections: 15
Relevant connections: ~3
Noise ratio: 90%+

⚠ INFORMATION OVERLOAD!

The  bank (money?)  by  the  river  bank (shore?)

Word Position

Problem:
• Which 'bank'?
• Too many signals
• Important info lost

# First Attempt: Connect Everything

**Early 2010s approach: Just connect everything!**



**Results:**

- √ Works for short sentences
- × Fails on long text
- × Can't distinguish important from noise

**What happens with full connections:**

**Step 1:** Every word becomes a vector

- "cat" $\rightarrow$ [0.7, 0.2, 0.5]
- "sat" $\rightarrow$ [0.3, 0.8, 0.4]
- "mat" $\rightarrow$ [0.6, 0.1, 0.7]

**Step 2:** Compute all dot products

- cat $\cdot$ sat $= 0.59$
- cat $\cdot$ mat $= 0.87$
- sat $\cdot$ mat $= 0.52$

**Step 3:** Average everything Result: Information soup!

| | cat | sat | mat |
|---|---|---|---|
| cat | 1.0 | 0.59 | 0.87 |
| sat | 0.59 | 1.0 | 0.52 |
| mat | 0.87 | 0.52 | 1.0 |

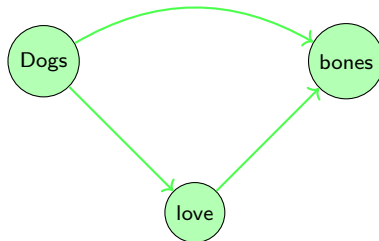All relationships computed but no focus!

# SUCCESS! (On Simple Cases)

**When it works:**

**Short, clear sentences:**
- "Dogs love bones" ✓
- "Paris is beautiful" ✓
- "Water is wet" ✓

**Why it works here:**
- Few connections (9 total for 3 words)
- All connections matter
- No ambiguity



**Clear signal!**

**Success Rate:** 95% on 3-5 word sentences

**When it fails:**

**Real-world sentence:** "The bank agreed to provide a loan for the house by the river bank after reviewing the application that the customer submitted last Tuesday."

**Problems:**

- 20 words = 400 connections!
- "bank" (financial) vs "bank" (river)
- Long-distance dependencies
- Most connections are noise



FAILURE: Signal Lost in Noise (Long Sentences)
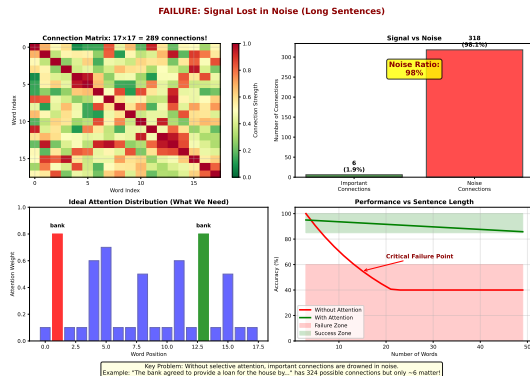
**Failure Rate:** 60% on 15+ word sentences
Signal drowned in noise!

**Eye-tracking studies reveal:**
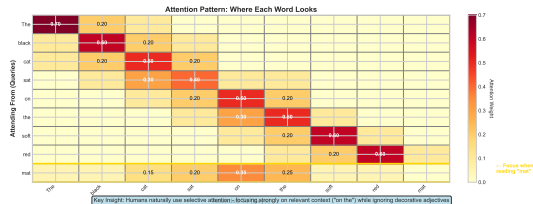
**Humans DON'T read every word equally!**

Reading: "The quick brown fox jumps"

- Focus on "fox" and "jumps"
- Skim "the" and "brown"
- Context determines focus

**Human attention is:**

- Selective (ignore irrelevant)
- Contextual (meaning-based)
- Efficient (focus on key parts)



When your eyes reach "mat", your brain focuses on:

Attention Pattern: Where Each Word Looks

**Key Insight:**
We need SELECTIVE attention,
not full connections!

**The Breakthrough (2017):**

Instead of connecting everything equally,
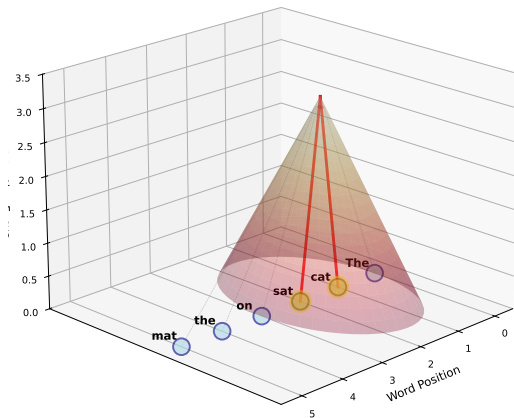**learn WHICH connections matter!**

**Attention Mechanism:**

1. Compute importance scores
2. Focus on high scores
3. Ignore low scores

**Like a spotlight:**

- Bright on important words
- Dim on filler words
- Adjustable based on context



Selective Attention: Spotlight on Important Words

Bright spotlight = High attention
Dim areas = Low attention
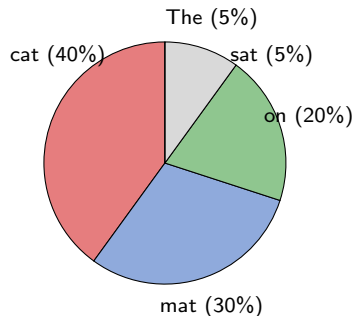
# Breaking It Down: Attention as Percentages

**Example:** "The cat sat on mat"

**When processing "sat":**
- 40% attention to "cat" (who sat?)
- 30% attention to "mat" (where?)
- 20% attention to "on" (relation)
- 5% to "The" (not important)
- 5% to itself

**These percentages:**
- Always sum to 100%
- Change for each word
- Learned from data



**Softmax** ensures percentages
always total 100%

**Measuring similarity with angles:**

**Dot Product Formula:**

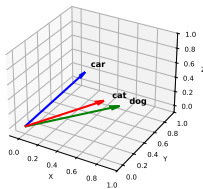$$\text{similarity} = \vec{A} \cdot \vec{B} = |\vec{A}| \times |\vec{B}| \times \cos(\theta)$$

**What this means:**
- Same direction: $\cos(0°) = 1$ (max similar)
- Perpendicular: $\cos(90°) = 0$ (unrelated)
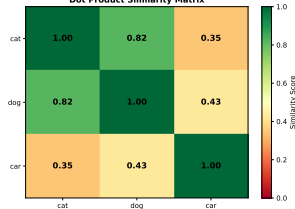- Opposite: $\cos(180°) = -1$ (opposite)

**Example:**
- cat $\cdot$ dog $= 0.8$ (similar animals)
- cat $\cdot$ car $= 0.1$ (very different)



Dot product = Semantic similarity

**For each word, we ask 3 questions:**

1. **Query (Q):** "What am I looking for?"
   - Cat's query: "Who performed action on me?"

2. **Key (K):** "What do I offer?"
   - Sat's key: "I am an action verb"

3. **Value (V):** "What information do I provide?"
   - Sat's value: "Past tense sitting action"

**Matching:** Q·K determines attention weight



Query-Key-Value: Three Different Perspectives on Same Word
Each transformation extracts different aspects of meaning

The same word "mat" is transformed into 3 spaces: Q asks what it needs, K advertises what it has, V provides the actual content
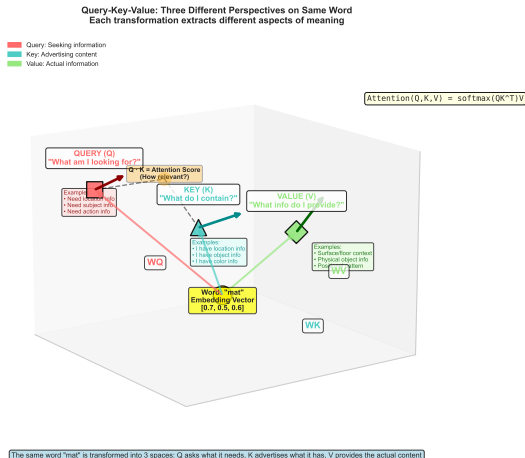
**QKV** transforms each word
into searcher, identifier, and content

**Query-Key-Value: Three Different Perspectives on Same Word**
**Each transformation extracts different aspects of meaning**



- Query: Seeking information
- Key: Advertising content
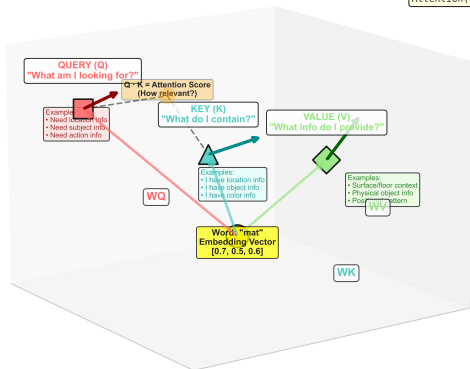- Value: Actual information

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

**QUERY (Q)**
"What am I looking for?"

Q · K = Attention Score
(How relevant?)

**KEY (K)**
"What do I contain?"

**VALUE (V)**
"What info do I provide?"

Examples:
• Need location info
• Need subject info
• Need action info

Examples:
• I have location info
• I have object info
• I have color info

Examples:
• Surface/floor context
• Physical object info
• Position pattern

WQ

WV

Word: "mat"
Embedding Vector
[0.7, 0.5, 0.6]

WK
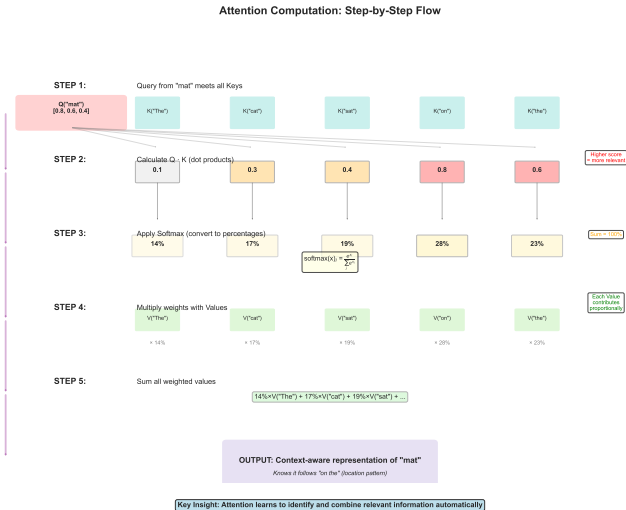
The same word "mat" is transformed into 3 spaces: Q asks what it needs, K advertises what it has, V provides the actual content

Query, Key, Value Transformation

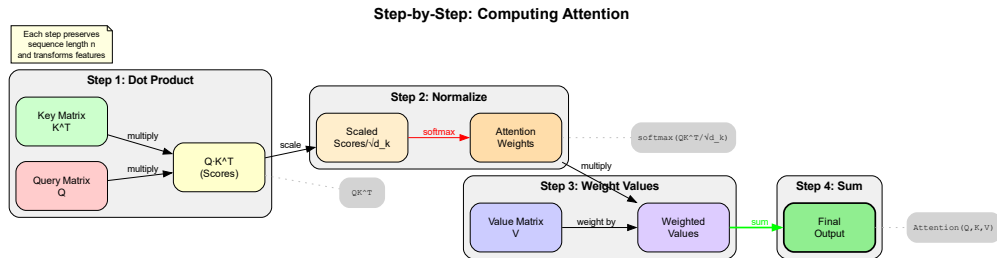**Attention Computation: Step-by-Step Flow**



**STEP 1:** Query from "mat" meets all Keys

Q("mat")
[0.3, 0.5, 0.6, 0.4]   K("The")   K("sat")   K("mat")   K("on")   K("The")

**STEP 2:** Calculate Q · K (dot products)

0.1   0.3   0.4   0.8   0.6

Higher score = more relevant

**STEP 3:** Apply Softmax (convert to percentages)

14%   17%   19%   28%   23%

$$\text{softmax}(x)_i = \frac{e^{x_i}}{\sum e^{x_j}}$$

Sum = 100%

**STEP 4:** Multiply weights with Values

V("The")   V("sat")   V("mat")   V("on")   V("The")

+ 14%   + 17%   + 19%   + 28%   + 23%

Each Value contributes proportionally

**STEP 5:** Sum all weighted values

14%×V("The") + 17%×V("sat") + 19%×V("mat") + ...

**OUTPUT: Context-aware representation of "mat"**
*Knows it follows "on the" (location pattern)*

Key Insight: Attention learns to identify and combine relevant information automatically

**1.** Q·K **2.** Softmax **3.** Weight V **4.** Sum

**Step-by-Step: Computing Attention**

# Multiple Perspectives: 4 Different Experts

**Multi-Head Attention:**

Like having 4 specialist readers:

**Head 1:** Grammar Expert
- Subject-verb agreement
- Sentence structure

**Head 2:** Meaning Expert
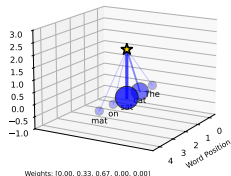- Semantic relationships
- Word meanings

**Head 3:** Position Expert
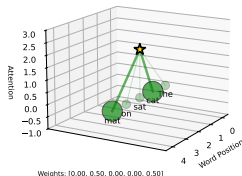- Word order
- Distance relationships

**Head 4:** Context Expert
- Broader context
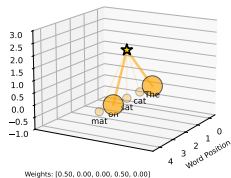- Document theme



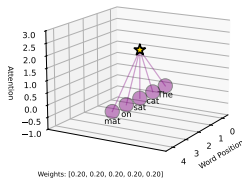Multi-Head Attention: 4 Different Perspectives

Head 1: Grammar Expert
Weights: [0.00, 0.33, 0.67, 0.00, 0.00]

Head 2: Meaning Expert
Weights: [0.00, 0.50, 0.00, 0.00, 0.50]

Head 3: Position Expert
Weights: [0.50, 0.00, 0.00, 0.50, 0.00]

Head 4: Context Expert
Weights: [0.20, 0.20, 0.20, 0.20, 0.20]

Each head learns different attention patterns:
- Grammar: Subject-verb relationships
- Meaning: Semantic connections

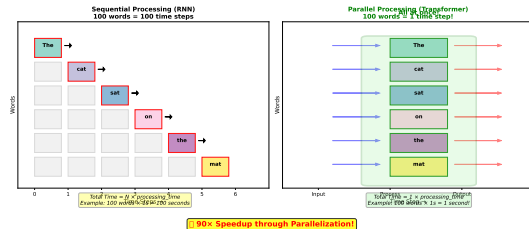# The Speed Revolution: Everything at Once

**Old way (RNN):** Sequential

- Process word 1, then 2, then 3...
- 100 words = 100 time steps
- Like reading one letter at a time

**New way (Transformer):** Parallel

- Process ALL words simultaneously
- 100 words = 1 time step!
- Like seeing whole page instantly

**Speed improvement:**

- Training: 90 days $\rightarrow$ 1 day
- Inference: 10 seconds $\rightarrow$ 0.1 seconds

## Preserving Order: Where Words Live

**Problem:** Parallel loses word order!

"Dog bites man" vs "Man bites dog"
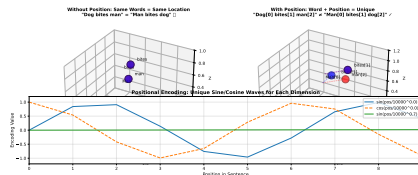
- Same words, different meaning
- Position matters!

**Solution: Positional Encoding**

- Add position information
- Use sine/cosine waves
- Different frequency for each dimension

**Like GPS for words:**

- Word + Position = Unique identity
- Model knows where each word is

> Position encoding =
> Word's address

# The Highway: Residual Connections

**Problem with deep networks:**
- Information gets lost/distorted
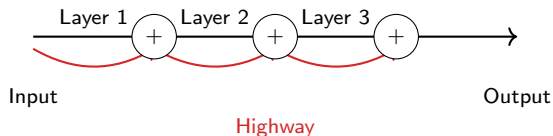- Like telephone game
- Each layer adds noise

**Solution: Skip Connections**
- Create "highways" for information
- Original signal preserved
- Add refinements, don't replace

**Formula:**

$$\text{Output} = \text{Layer}(x) + x$$

Always keep original, add improvements



Residual = Original + Refinement
Never lose information!

**The Complete Architecture:**

1. **Input:** Words $\rightarrow$ Embeddings + Position

2. **Attention Block:**
   - Multi-head attention
   - Add & normalize (residual)

3. **Feed-Forward Block:**
   - Two linear layers
   - Add & normalize (residual)

4. **Stack 12 times** (BERT) or 96 times (GPT-3)

5. **Output:** Next word prediction
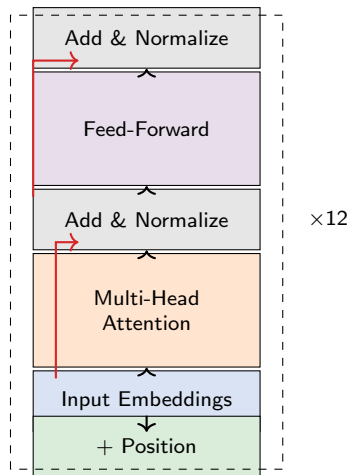
## Proof It Works: Real Results

**Benchmark Results (2017-2024):**

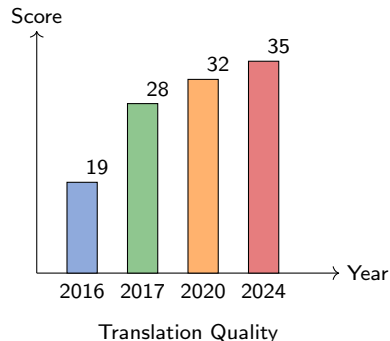**Translation (BLEU scores):**
- 2016 (RNN): 19.2
- 2017 (Transformer): 28.4 ✓
- 2024 (GPT-4): 35.1 ✓

**Question Answering:**
- Human performance: 89%
- BERT (2018): 93% ✓
- GPT-4 (2023): 96% ✓

**Training Speed:**
- RNN: 3 months
- Transformer: 1 day ✓



Translation Quality

**Transformers:** State-of-the-art
on EVERY language task!

**Timeline of Breakthroughs:**

**2017:** Transformer paper
- "Attention is All You Need"
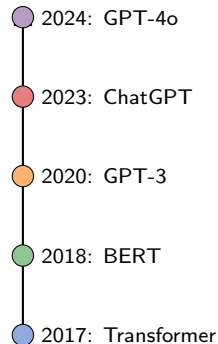- 8 researchers at Google

**2018:** BERT
- Bidirectional understanding
- Crushed 11 benchmarks

**2020:** GPT-3
- 175 billion parameters
- Few-shot learning

**2023:** ChatGPT/GPT-4
- Conversation ability
- 100M users in 2 months

2024: GPT-4o

2023: ChatGPT

2020: GPT-3

2018: BERT

2017: Transformer

7 years: From research paper
to 1 billion users!

## Practice Problems

**Problem 1: Calculate Attention**
Given vectors:

- Query: [1, 0, 1]
- Key1: [1, 1, 0]
- Key2: [0, 1, 1]
- Key3: [1, 0, 1]

Calculate:

1. $Q \cdot K$ for each key
2. Apply softmax
3. Which word gets most attention?

**Problem 2: Multi-Head Design**
Design 3 attention heads for: "The bank near the river bank"
What should each head focus on?

**Problem 3: Architecture**
Draw transformer architecture for:

- 2-layer transformer
- Show residual connections
- Label each component

**Problem 4: Complexity**
For a 100-word sentence:

- How many attention scores?
- Memory requirement?
- Why is this $O(n^2)$?

**Solutions:** Available after lab session

# Summary: The Three Core Principles

**1. Parallel Processing**
- All words processed simultaneously
- 90x faster than sequential
- Enables large-scale training
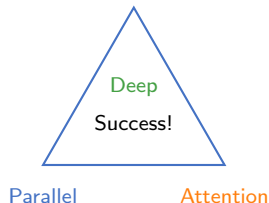
**2. Attention Mechanism**
- Focus on relevant connections
- Learn what matters from data
- Multiple perspectives (heads)

**3. Deep Architecture**
- Stack many layers (12-96)
- Residual connections preserve info
- Each layer refines understanding

**TRANSFORMERS**

Deep

Success!

Parallel          Attention

These 3 principles revolutionized AI
and created ChatGPT, BERT, and more!

# Where You Use Transformers Every Day

**Search Engines:**

- Google Search (BERT)
- Autocomplete suggestions
- "Did you mean...?"

**Translation:**

- Google Translate
- Real-time captions
- Document translation

**Writing Assistants:**

- Grammarly corrections
- Email suggestions (Gmail)
- Code completion (Copilot)

**Virtual Assistants:**

- ChatGPT conversations
- Siri/Alexa understanding
- Customer service bots

**Content Creation:**

- Image generation (DALL-E)
- Video subtitles
- Article summaries

**Fact:** You interact with transformers
dozens of times daily!

## Check Your Understanding

**Quick Quiz:**

**Q1:** Why are transformers fast?

A) Smaller models

B) Parallel processing ✓

C) Better hardware

D) Simpler math

**Q2:** What does attention do?

A) Adds more parameters

B) Focuses on relevant words ✓

C) Speeds up training

D) Reduces memory

**Q3:** Purpose of residual connections?

A) Preserve information ✓

B) Add complexity

C) Reduce size

D) Speed up inference

**Can you now:**

☐ Explain word embeddings?

☐ Calculate dot product similarity?

☐ Describe Query, Key, Value?

☐ Draw attention mechanism?

☐ List transformer components?

☐ Explain parallel vs sequential?

**Congratulations!**
You understand the technology
behind ChatGPT!
From zero to transformer expert
in 27 slides!

## Appendix A: Mathematical Foundations

**Formal Attention Equation:**

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

**Where:**

- $Q \in \mathbb{R}^{n \times d_k}$: Queries
- $K \in \mathbb{R}^{n \times d_k}$: Keys
- $V \in \mathbb{R}^{n \times d_v}$: Values
- $n$: Sequence length
- $d_k$: Key/Query dimension
- $\sqrt{d_k}$: Scaling factor

**Why Scale by $\sqrt{d_k}$?**

- Dot products grow with dimension
- Large values $\rightarrow$ softmax saturation
- Scaling keeps gradients stable

**Matrix Shapes Example:**

- Input: $[10, 512]$ (10 words)
- $Q, K, V$: $[10, 64]$ each
- Attention scores: $[10, 10]$
- Output: $[10, 64]$

**Key Insight:** The attention matrix is $n \times n$ - quadratic in sequence length!

# Appendix B: Multi-Head Attention Mathematics

**Multi-Head Formula:**

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$$

**Where each head:**

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

**Parameter Matrices:**

- $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$
- $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$
- $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$
- $W^O \in \mathbb{R}^{hd_v \times d_{model}}$

**Typical Dimensions:**

- $d_{model} = 512$
- $h = 8$ heads
- $d_k = d_v = d_{model}/h = 64$
- Total parameters: $4 \times d_{model}^2$

**Computational Benefits:**

- Heads run in parallel
- Different representation subspaces
- Similar cost to single-head
- Better performance in practice

8 heads $\times$ 64 dimensions = 512 total dimensions (preserved)

## Appendix C: Positional Encoding Mathematics

**Sinusoidal Position Encoding:**

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

**Where:**

- *pos*: Position in sequence (0, 1, 2, ...)
- *i*: Dimension index
- $d_{model}$: Model dimension (e.g., 512)

**Properties:**

- Unique encoding per position
- Smooth variation across positions
- Can extrapolate to longer sequences

**Why This Formula?**

- Relative positions preserved
- $PE_{pos+k}$ can be represented as linear function of $PE_{pos}$
- Different frequencies per dimension
- No learned parameters needed

**Wavelengths:**

- Dimension 0-1: wavelength $2\pi$
- Dimension 510-511: wavelength $2\pi \cdot 10000$
- Covers short to long-range dependencies

Position encoding added to word embedding: $x' = x + PE_{pos}$

## Appendix D: Complexity Analysis

**Self-Attention Complexity:**
- Time: $O(n^2 \cdot d)$
- Space: $O(n^2 + n \cdot d)$
- Attention matrix: $n \times n$

**RNN Complexity:**
- Time: $O(n \cdot d^2)$ sequential
- Space: $O(d)$ per step
- Must process sequentially

**Comparison (n=100, d=512):**
- Transformer: $100^2 \times 512 = 5.12M$ ops
- RNN: $100 \times 512^2 = 26.2M$ ops
- But RNN is sequential!

**Parallelization Benefits:**
- All positions computed simultaneously
- GPU utilization: nearly 100%
- Training speedup: 50-100x
- Inference speedup: 10-20x

**Memory Requirements:**
- Attention scores: $O(n^2)$ bottleneck
- Max sequence typically 512-2048
- Recent work on sparse attention
- Linear attention variants emerging

**Tradeoff:** Quadratic memory for massive parallelization gain

## Appendix E: Implementation Details

**Layer Normalization:**

$$\text{LayerNorm}(x) = \gamma \frac{x - \mu}{\sigma} + \beta$$

Applied as: $\text{LayerNorm}(x + \text{Sublayer}(x))$

**Dropout Positions:**

- After each sublayer: 0.1
- Attention weights: 0.1
- Embedding layer: 0.1

**Feed-Forward Network:**

$$\text{FFN}(x) = \text{ReLU}(xW_1 + b_1)W_2 + b_2$$

- Hidden size: $4 \times d_{model}$
- Two linear transformations
- ReLU activation between

**Learning Rate Schedule:**

$$lr = d_{model}^{-0.5} \cdot \min(step^{-0.5}, step \cdot warmup^{-1.5})$$

**Typical Hyperparameters:**

- $d_{model} = 512$ or $768$
- $h = 8$ or $12$ heads
- $d_{ff} = 2048$ or $3072$
- Layers: 6 (base) or 12 (large)
- Warmup steps: 4000
- Batch size: 4096 tokens

**Training Tips:**

- Label smoothing: 0.1
- Gradient clipping: 1.0
- Adam optimizer: $\beta_1 = 0.9, \beta_2 = 0.98$

**Total Parameters (Base):** 65M for 6-layer transformer