Attention & the Transformer Architecture
Multimodal Computational Methods in Political Science

Computational Social Science Program [1]

[1]LMU Munich — Computational Social Science MA Program

June 2026

Course Roadmap

1. Text as Data: Foundations & Preprocessing
2. Classical Text Classification
3. Word Embeddings & Vector Spaces
4. Document Representations & Topic Models
5. Neural Networks & Sequence Models
6. **Attention & the Transformer Architecture** ← *Today*
7. Transfer Learning & Fine-Tuning BERT

Today we meet the architecture that transformed NLP since 2017: the **Transformer**. The mechanism at its heart — **self-attention** — finally solves the three limitations of RNNs from last week. By the end of today, you'll understand how BERT, GPT, and every modern language model are built.

Today's Outline

Part I: From RNNs to Attention

- Recap: the three limits of LSTMs
- The intuition behind attention
- "Attention is All You Need" (2017)

Part II: Self-Attention Mechanics

- Query, Key, Value
- Scaled dot-product attention
- Worked example
- Why scaling matters

Part III: Multi-Head Attention

- Why multiple heads?
- What different heads learn

Part IV: The Transformer Block

- Positional encodings (the missing piece)
- Residual connections & LayerNorm
- Feed-forward sub-layers
- The full encoder block
- Encoder vs. decoder vs. encoder-only

Part V: Why This Matters

- Parallelism & scale
- Political science applications

Part VI: Practical (Python)

- Visualizing attention in BERT

Table of Contents

1. Part I: From RNNs to Attention
2. Part II: Self-Attention Mechanics
3. Part III: Multi-Head Attention
4. Part IV: The Transformer Block
5. Part V: Why This Matters
6. Part VI: Practical Session
7. Wrap-up

Recap: The Three Limits of LSTMs

LSTMs were the workhorse of NLP from 2015–2018. They solved the vanishing gradient problem and enabled real progress on translation, NER, and question answering. But they have three fundamental limits.

1. **No parallelism.** The hidden state at step t depends on the hidden state at step $t-1$. You cannot compute steps in parallel. Modern GPUs are wasted.
2. **Long-range dependencies remain hard.** Information from word 1 must pass through 499 cell-state updates to reach word 500. Some signal still gets lost.
3. **Information bottleneck.** The cell state is a fixed-size vector. The entire history must be compressed into it, and longer sequences lose more.

The wish list

We want an architecture that:

- Processes all positions **in parallel**
- Lets any position directly access any other position (no bottleneck)
- Captures long-range dependencies without exponentially decaying signal

The answer: self-attention.

"Attention is All You Need" (2017)

In June 2017, eight researchers at Google published a paper with a deliberately provocative title: **Attention Is All You Need** (Vaswani et al.).

The bold claim: you can throw away all the recurrence (RNNs, LSTMs) and convolution. Build a model out of **nothing but attention layers**. It will work better, train faster, and scale further.

They were right. The Transformer architecture they introduced is now the foundation of:

- **BERT** (2018) — bidirectional encoder, the basis of most political-science NLP today
- **GPT-2, GPT-3, GPT-4** (2019–2023) — decoder-only, ChatGPT and friends
- **T5, BART** — encoder-decoder models for translation, summarization
- Every modern LLM: LLaMA, Mistral, Claude, Gemini, ...

Key Concept

Understanding the Transformer is the conceptual key to everything that has happened in NLP since 2017. Today we build it from the ground up.

The Intuition: "Look at What Matters"

When humans read a sentence, we don't process word by word in isolation. Our brain unconsciously links each word to other relevant words.

Reading a political sentence

"The Prime Minister, despite vocal opposition from her own party, announced new climate measures."

When you read "announced," your attention drifts back to "Prime Minister" (who announced?). When you read "her," your brain points to "Prime Minister" again (whose party?). When you read "measures," your attention links forward to "climate" (what kind?).

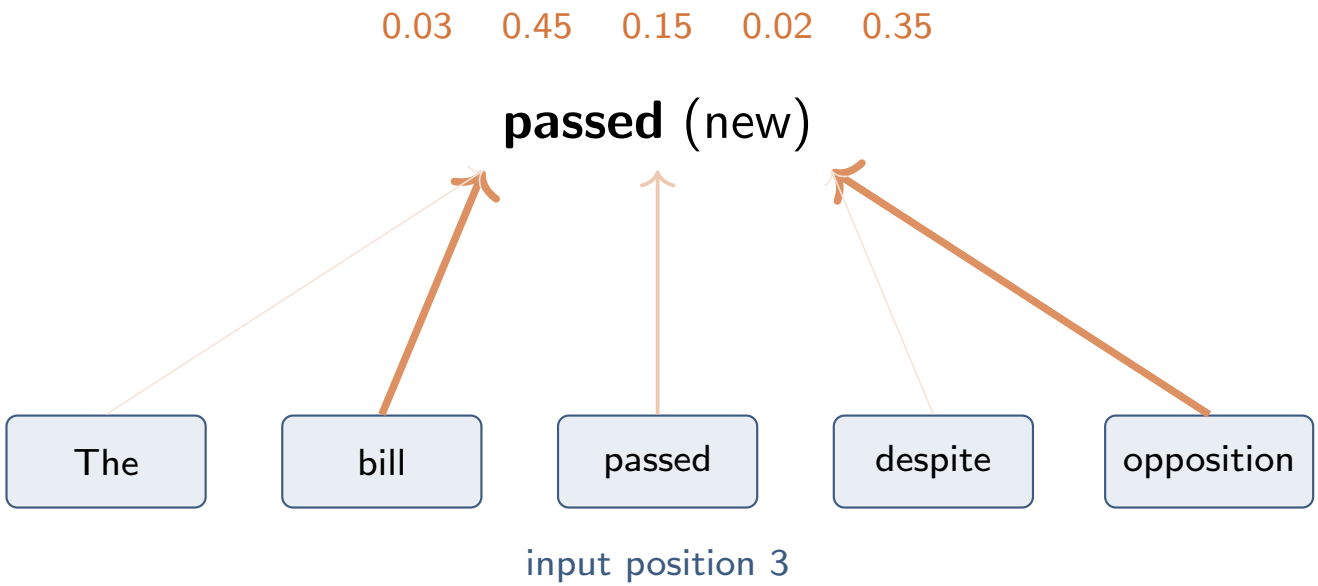
Attention is this mechanism, made mathematical. For each word in the input, the model:

1. Asks: *which other words in this sentence are relevant to me?*
2. Computes a weight for each other word (high for relevant, low for irrelevant)
3. Builds its own representation as a weighted combination of all the other words

Crucially: every word can attend to every other word **directly**, in a single step. No sequential walk-through. No fixed-size bottleneck.

Self-Attention: A Picture

Each word produces a new representation by "mixing" information from all other words, weighted by relevance.



What you see:

- Output for "passed" is a weighted sum of all input positions
- Heavy weights (thick arrows) on *bill* and *opposition* — semantically related
- Light weights on function words like *the* and *despite*
- The model **learns** these weights from data, for each pair of positions

Table of Contents

1. Part I: From RNNs to Attention
2. Part II: Self-Attention Mechanics
3. Part III: Multi-Head Attention
4. Part IV: The Transformer Block
5. Part V: Why This Matters
6. Part VI: Practical Session
7. Wrap-up

The Q, K, V Framework

For each input position, attention uses three derived vectors:

Query ($\mathbf{Q}$)	Key ($\mathbf{K}$)	Value ($\mathbf{V}$)
What am I looking for? The query represents this position's "question" to the rest of the sequence.	What do I offer? The key represents what information this position makes available to others.	What will I give? The value is the actual content this position contributes, if attended to.

The library analogy:

- You walk into a library (the sequence) with a *question* (your query).
- Each book has a *label* on the spine (its key) and *content* inside (its value).
- You compare your question to every label. The labels that match best get the most attention.
- You read the matched books, weighted by how well their labels matched.

Each of Q, K, V is a vector. They are all derived from the same input by separate **learned** linear transformations: $\mathbf{Q} = \mathbf{X}\mathbf{W}_Q$, $\mathbf{K} = \mathbf{X}\mathbf{W}_K$, $\mathbf{V} = \mathbf{X}\mathbf{W}_V$. The weight matrices are what the model learns.

Computing Attention: The Steps

For one position i attending over the sequence:

1. **Score** the query at position i against the keys at every position j:

$$s_{ij} = \mathbf{q}_i \cdot \mathbf{k}_j \qquad \text{(dot product)}$$

   Big positive score $\rightarrow$ key matches query well.
   Big negative score $\rightarrow$ poor match.

2. **Scale** the scores by $\sqrt{d_k}$ (where d_k is the dimensionality of the keys). We'll explain why shortly.

$$\tilde{s}_{ij} = \frac{s_{ij}}{\sqrt{d_k}}$$

3. **Softmax** to turn scores into a probability distribution:

$$\alpha_{ij} = \frac{e^{\tilde{s}_{ij}}}{\sum_{j'} e^{\tilde{s}_{ij'}}}$$

   The α_{ij} are the **attention weights**: how much position i attends to position j. They sum to 1 across j.

4. **Weighted sum** of values:

$$\text{output}_i = \sum_j \alpha_{ij} \mathbf{v}_j$$

The Whole Thing in One Formula

Putting it all together — attention in matrix form:

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

Breaking it down:

- $\mathbf{Q}\mathbf{K}^\top$ — all pairwise dot products between queries and keys at once. If you have n positions, this is an $n \times n$ matrix of scores.
- Divide by $\sqrt{d_k}$ — scaling (next slide).
- Apply softmax **row-wise** — each row becomes a probability distribution.
- Multiply by $\mathbf{V}$ — weighted sum of values for each position.

This single formula — "scaled dot-product attention" — is the heart of every modern language model.

Computed in parallel for all positions simultaneously. **No sequential walk-through.**

Worked Example (1/2): The Setup

Sentence: "*The bill passed*" (3 tokens, for simplicity). Suppose after the linear projections, our 3 tokens have these (toy) 2-dimensional Q, K, V vectors:

Token	Query $\mathbf{q}$		Key $\mathbf{k}$		Value $\mathbf{v}$	
The	1.0	0.0	0.2	0.0	0.1	0.2
bill	0.0	1.0	0.1	0.9	0.7	0.3
passed	0.5	0.5	0.7	0.5	0.4	0.5

Goal: compute the attention output for "passed" (position 3). **Its query is** $\mathbf{q}_3 = [0.5, 0.5]$. We will:

1. Score $\mathbf{q}_3$ against each key
2. Scale by $\sqrt{d_k} = \sqrt{2} \approx 1.414$
3. Softmax
4. Take weighted sum of values

Worked Example (2/2): The Math

Step 1: Dot product scores

$$s_{3,1} = \mathbf{q}_3 \cdot \mathbf{k}_{\text{The}} = 0.5 \times 0.2 + 0.5 \times 0.0 = 0.10$$
$$s_{3,2} = \mathbf{q}_3 \cdot \mathbf{k}_{\text{bill}} = 0.5 \times 0.1 + 0.5 \times 0.9 = 0.50$$
$$s_{3,3} = \mathbf{q}_3 \cdot \mathbf{k}_{\text{passed}} = 0.5 \times 0.7 + 0.5 \times 0.5 = 0.60$$

Step 2: Scale by $\sqrt{2} \approx 1.414$**:** $\tilde{s} \approx [0.071,\ 0.354,\ 0.424]$

Step 3: Softmax (now non-uniform):

$$e^{0.071} \approx 1.073, \quad e^{0.354} \approx 1.424, \quad e^{0.424} \approx 1.528 \quad (\text{sum} \approx 4.025)$$

$$\alpha_3 \approx [0.267,\ 0.354,\ 0.379]$$

Step 4: Weighted sum of values

$$\text{output}_3 = 0.267 \cdot [0.1, 0.2] + 0.354 \cdot [0.7, 0.3] + 0.379 \cdot [0.4, 0.5] \approx [0.426,\ 0.349]$$

"Passed" attends most strongly to itself (0.379) and to "bill" (0.354), least to "the" (0.267) — the mechanism gives more weight to semantically related tokens. In real BERT, learned Q/K matrices would produce much sharper patterns than this toy example.

Why Divide by $\sqrt{d_k}$?

The scaling factor $\sqrt{d_k}$ is not arbitrary. It exists for one reason: to keep softmax well-behaved.

The problem without scaling:
In high-dimensional spaces (say $d_k = 64$), random dot products tend to have large magnitudes. If scores are very large in absolute value, softmax becomes "peaky" — one position gets attention weight ≈ 1 and all others get ≈ 0.

Consequence: gradients flowing through softmax become tiny (most weights are nearly 0 or 1, derivatives are flat). Training stalls.

The fix: divide by $\sqrt{d_k}$. The variance of the dot product of two random vectors of dimension d_k is proportional to d_k; dividing by $\sqrt{d_k}$ keeps the standard deviation roughly constant.

Key Concept

The scaling is a numerical-stability trick. You don't need to remember the derivation. Just know: **without it, training is unstable in higher dimensions.**

Attention Is Permutation-Equivariant

An important property of self-attention: if you reorder the input tokens, the outputs reorder correspondingly — but **the content doesn't change**.

In other words: self-attention doesn't know which word came first!

The order-blindness problem

"The opposition supported the bill"
"The bill supported the opposition"

These have the same words. Self-attention, on its own, produces the same set of token representations for both — just in a different order. The meaning is completely different, but attention cannot tell them apart.

Caution

This is a fatal problem if left unsolved. Almost everything in language depends on word order: subject vs. object, negation scope, syntactic structure.
The fix: **positional encodings**. We'll get to them in Part IV.

Table of Contents

1. Part I: From RNNs to Attention
2. Part II: Self-Attention Mechanics
3. Part III: Multi-Head Attention
4. Part IV: The Transformer Block
5. Part V: Why This Matters
6. Part VI: Practical Session
7. Wrap-up

One Head Is Not Enough

The problem with single-head attention: different kinds of relationships in language need different attention patterns.

Different relationships, same sentence

"The Prime Minister, despite vocal opposition, announced climate measures."

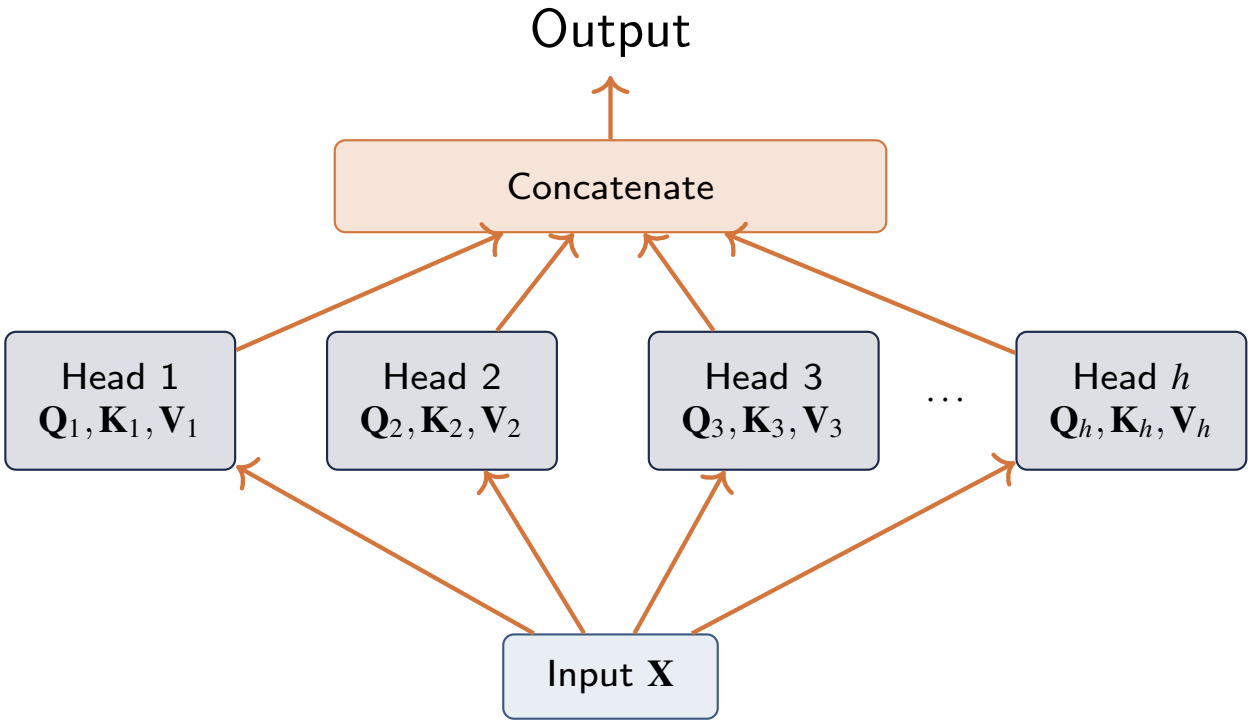
For the word "announced," a useful model might want to attend to:

- **Subject relations**: "Prime Minister" (who announced?)
- **Object relations**: "measures" (announced what?)
- **Contrast/concession**: "despite" and "opposition"
- **Semantic field**: "climate" (topical)

A single set of attention weights cannot capture all four simultaneously.

The solution: multiple "attention heads" running in parallel. Each head learns its own Q, K, V projections and attends differently. Their outputs are concatenated.

Multi-Head Attention: The Architecture



Each head: smaller dimension ($d_k = d_{\text{model}}/h$), own learned $\mathbf{W}_Q$, $\mathbf{W}_K$, $\mathbf{W}_V$, own attention computation.

Then concatenate all h outputs and apply one final linear projection to mix them.

BERT-base uses 12 heads of dimension 64 each ($12 \times 64 = 768$). BERT-large uses 16 heads of 64.

What Different Heads Learn

Empirical finding: different heads specialize in different relationships — often interpretable ones.

Patterns found in BERT (Clark et al. 2019, "What Does BERT Look At?"):

- **Some heads attend to the next/previous word** (positional patterns).
- **Some heads attend to syntactic dependencies**: direct objects to their verbs, possessive pronouns to their referents.
- **Some heads attend to delimiters** ([SEP], [CLS]) — a kind of "default" position when no other word is informative.
- **Some heads attend to coreference**: pronouns to the entities they refer to.
- **Some heads remain hard to interpret** — they encode something useful, but not human-readable.

Useful for political-science research

You can **visualize attention** in pre-trained models to study how they process political text. This is a growing research area: probing model biases, comparing how a model processes left- vs. right-wing rhetoric, etc.
We'll do this in today's practical session.

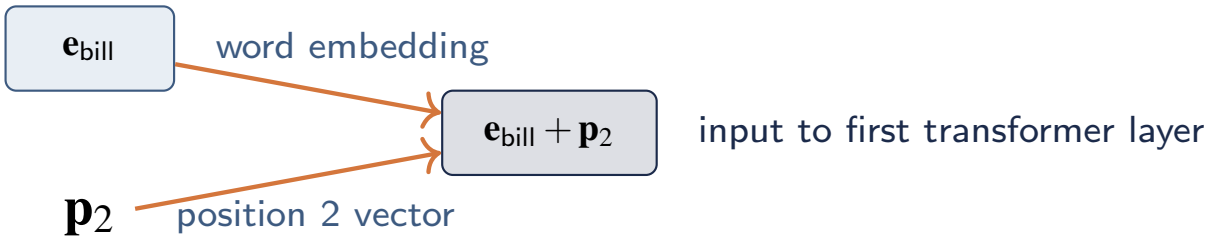
Table of Contents

1. Part I: From RNNs to Attention
2. Part II: Self-Attention Mechanics
3. Part III: Multi-Head Attention
4. Part IV: The Transformer Block
5. Part V: Why This Matters
6. Part VI: Practical Session
7. Wrap-up

Positional Encodings: The Missing Piece

Self-attention doesn't know word order. We must **inject** positional information into the inputs before attention.

The idea: add a position-dependent vector to each word's embedding.



Two main flavors:

- **Sinusoidal** (original Transformer): fixed functions using sines and cosines at different frequencies. Mathematically elegant; generalizes to unseen sequence lengths.
- **Learned** (BERT, most modern models): just another learned vector per position. Simpler. Limited to the max length seen during training.

You don't need to memorize the sinusoidal formulas. The key insight: position is encoded as a vector and added to the word embedding. After this step, the model knows where each token is.

Residual Connections and Layer Normalization

Stacking many attention layers is hard. Two tricks make training deep transformers possible.

Residual (skip) connections
Add the input of a sub-layer to its output:

$$\mathbf{y} = \mathbf{x} + \text{SubLayer}(\mathbf{x})$$

Why: gradients can flow "around" the sub-layer through the addition. Even if the sub-layer makes things worse, you can fall back to the identity.

This idea, from ResNets (2015), is essential for training networks dozens or hundreds of layers deep.

Layer normalization
Re-center and rescale each token's vector so it has mean 0 and standard deviation 1 (with learned scale and shift):

Why: keeps the magnitudes of activations stable across layers. Prevents some from blowing up or vanishing during training.

The combination "*add & norm*" appears around every sub-layer in the transformer block.

These are engineering details, not deep conceptual ideas. But without them, the model would not train.

The Feed-Forward Sub-Layer

After attention (and add-&-norm), each Transformer block applies a small feed-forward network — **independently** to each token position.

$$\text{FFN}(\mathbf{x}) = \text{ReLU}(\mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\,\mathbf{W}_2 + \mathbf{b}_2$$

Two key points:

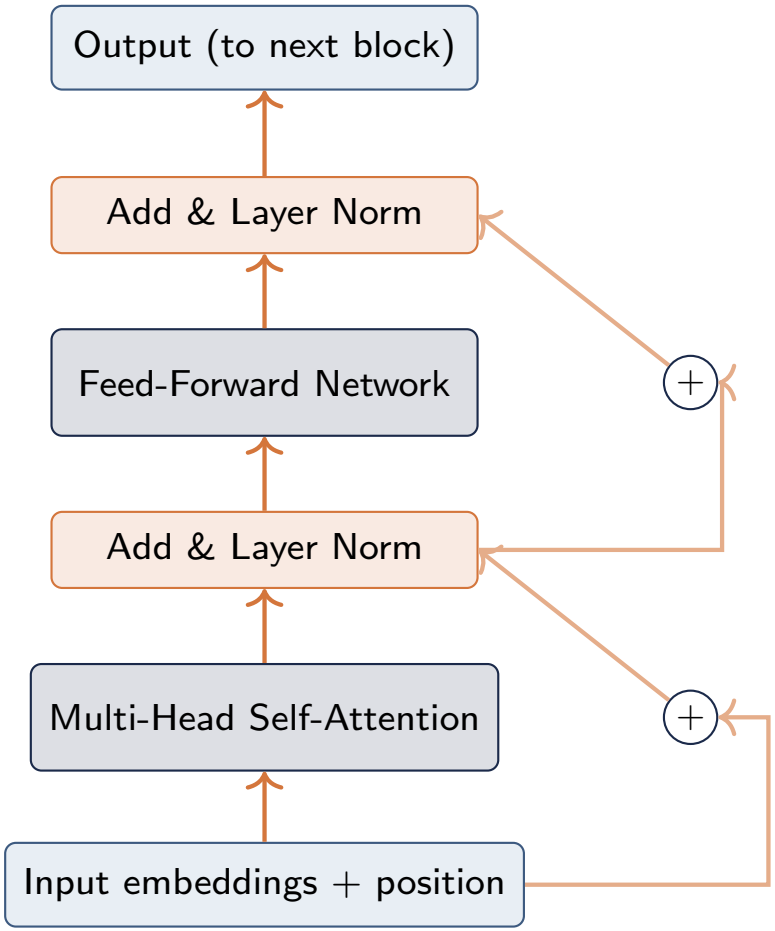
- It's the same FFN applied to every position separately (same weights, different inputs).
- It's wider than the model dimension: typically $4\times$ wider (e.g., $768 \to 3072 \to 768$).

What does it do? Two perspectives:

- Mathematically: applies non-linear transformation to each token's representation after the linear attention step.
- Intuitively: "processing" each token's information after it has gathered context from others.

In a typical Transformer, most of the parameters live in the FFNs, not in the attention.

The Full Transformer Encoder Block



One encoder block = attention + add&norm + FFN + add&norm.

A full encoder (e.g., BERT) stacks 12–24 such blocks on top of each other.

Each block transforms representations: earlier blocks pick up local/surface patterns, later blocks pick up abstract/semantic structure.

Three Transformer Families

The original Transformer (Vaswani et al. 2017) had an encoder *and* a decoder for machine translation. Today, three families dominate:

Encoder-only

Example: BERT, RoBERTa
Task: understanding text
Bidirectional attention: every token sees every other token.
Good for: classification, NER, similarity, sentence pairs.
This is what we'll fine-tune at Lecture7.

Decoder-only

Example: GPT family, LLaMA
Task: generating text
Causal attention: each token can only see itself and the past.
Good for: text generation, chatbots, code completion.
The architecture behind ChatGPT, Claude, etc.

Encoder-decoder

Example: T5, BART
Task: text $\rightarrow$ text
Encoder reads input, decoder generates output.
Cross-attention links them.
Good for: translation, summarization, structured QA.

For political science classification tasks, encoder-only (BERT-family) is usually the right choice — the most cost-effective, the easiest to fine-tune on limited compute, and the best-validated in the social-science literature.

Table of Contents

1. Part I: From RNNs to Attention
2. Part II: Self-Attention Mechanics
3. Part III: Multi-Head Attention
4. Part IV: The Transformer Block
5. Part V: Why This Matters
6. Part VI: Practical Session
7. Wrap-up

Transformers vs. LSTMs: The Comparison

	LSTM (Lecture 5)	**Transformer (today)**
Processing	Sequential (step by step)	Parallel (all positions at once)
Long-range dependencies	Decay over many steps	Direct: any-to-any in one layer
Memory bottleneck	Fixed-size cell state	No bottleneck; full pairwise attention
Training speed (per epoch)	Slow on GPU	Fast (parallelism!)
Scales to long sequences?	Yes, but quality drops	Quality holds; but $\mathcal{O}(n^2)$ **memory**
Inductive bias for sequences	Built-in (recurrence)	Must be learned (via positions)
State-of-the-art today?	Mostly historical	Yes — everywhere

Caution

The quadratic cost. Attention computes $n \times n$ pairwise scores. For a 10,000-word document, that's 100 million pairs — runs out of memory on a normal GPU. "Long-context" transformers (e.g., Longformer, BigBird) use sparse attention to scale to long inputs. This is an active research area.

Why Transformers Won: Scale

The Transformer's parallel structure means it can train on much more data than RNNs. This is the single biggest reason it won.

A scaling timeline:

- **2018: BERT-base** — 110M parameters, trained on 3B words.
- **2019: GPT-2** — 1.5B parameters, 40 GB of web text.
- **2020: GPT-3** — 175B parameters, ∼500B tokens.
- **2023: GPT-4, Claude, etc.** — estimated ∼1 trillion parameters, trillions of tokens.

The "scaling laws" lesson: model performance keeps improving as you make the model bigger, the data bigger, and the compute bigger. No plateau in sight (so far).

Implications for political science:

- Cutting-edge models are out of reach to train — but *not* to **use**.
- Pre-trained models can be downloaded and applied to your data with minimal compute.
- This is what we'll do in Lecture 7 with BERT.

Political Science Applications of Transformers

Where transformer-based methods are changing political science research:

- **High-quality classification with little data.** Fine-tuning a pre-trained model on a few hundred labeled examples often beats logistic regression with thousands. Especially useful for nuanced tasks (stance, framing, irony).
- **Multilingual analysis.** Multilingual BERT and XLM-RoBERTa make cross-country political text comparable without manual translation.
- **Zero-shot and few-shot classification.** Large language models can classify political text from natural-language prompts, with little or no training data. (Strong but unreliable — always validate.)
- **Probing political biases in models.** Models reproduce biases in their training data. Studying which biases they have, and how strongly, is itself a research question.
- **Contextual embeddings.** Unlike word2vec, BERT gives a different vector for every occurrence of a word. "Sovereignty" in a UKIP speech has a different vector than "sovereignty" in a Labour speech — measurable polarization.

All of this becomes hands-on in Lecture 7. Today: understanding the architecture; then fine-tuning it.

Table of Contents

1. Part I: From RNNs to Attention
2. Part II: Self-Attention Mechanics
3. Part III: Multi-Head Attention
4. Part IV: The Transformer Block
5. Part V: Why This Matters
6. Part VI: Practical Session
7. Wrap-up

Practical: Visualizing Attention in BERT

Inspecting What BERT Pays Attention To

What we'll do:

1. Load a pre-trained BERT model from Hugging Face
2. Feed it a UK parliamentary sentence
3. Extract the attention weights from each layer and each head
4. Visualize them as heatmaps
5. Compare attention patterns across layers

This is not yet fine-tuning — we're just looking inside a pre-trained model. Think of it as "dissecting" the architecture we just studied.

Tools:

- `transformers` (Hugging Face)
- `bertviz` (or `matplotlib` for heatmaps)
- Google Colab gives free GPU access if needed

Practical: What to Look For

Run the code in the practical and explore. Look for these patterns:

1. **Diagonal heads.** Some heads attend mostly to the same position (the diagonal). These "self-loop" heads carry information forward locally.
2. **Shifted-diagonal heads.** Some heads attend mostly to the previous or next token (off-diagonals). These capture local order.
3. **Vertical stripes on [CLS] or [SEP].** Many heads attend strongly to the special tokens — a kind of "default" attention.
4. **Subject–verb attention.** Look for verbs attending to subjects, especially in later layers (8–12).
5. **Coreference.** Pronouns attending to their referents ("her" $\rightarrow$ "Minister").
6. **Politically meaningful patterns.** What does the model do with "opposition" — does it attend to "despite"? Does "measures" attend to "climate"?

Try at least 3 different (layer, head) combinations. Save the most interesting heatmaps for the discussion.

Practical: Discussion Questions

1. Did you find a head whose pattern is interpretable as a linguistic relationship (subject–verb, coreference, etc.)?
2. How do attention patterns change between **early** layers (1–4) and **late** layers (9–12)?
3. The [CLS] token attracts a lot of attention in many heads. Why might that be? (Hint: think about how BERT is used for classification.)
4. Try a sentence with negation: *"The bill did not pass."* Where does "pass" attend? Does any head attend strongly to "not"?
5. Try a sentence with two parties: *"Labour rejected the Conservative proposal on immigration."* How does "rejected" attend?

Limit of interpretation: the model doesn't "mean" to attend in any particular way. Attention patterns are emergent statistics from training. They are often *suggestive* of linguistic structure, but they are not definitive explanations of model behavior.

Table of Contents

1. Part I: From RNNs to Attention
2. Part II: Self-Attention Mechanics
3. Part III: Multi-Head Attention
4. Part IV: The Transformer Block
5. Part V: Why This Matters
6. Part VI: Practical Session
7. Wrap-up

Key Takeaways

1. **Self-attention** lets every position in a sequence directly attend to every other position, fixing the three weaknesses of RNNs (no parallelism, bottleneck, decay).
2. **The Q, K, V framework** is the heart of attention: each position projects to a query, a key, and a value; queries match keys to determine how much to attend to each value.
3. **Scaled dot-product attention** $= \text{softmax}(\mathbf{QK}^\top/\sqrt{d_k})\mathbf{V}$. Memorize this formula — it's the single most important equation in modern NLP.
4. **Multi-head attention** runs several attentions in parallel, letting the model capture different relationships at once.
5. **Positional encodings** are essential — without them, attention is order-blind.
6. **A Transformer block** = multi-head attention + add&norm + feed-forward + add&norm, stacked many times.
7. **Three families** of Transformers: encoder-only (BERT, for understanding), decoder-only (GPT, for generation), encoder-decoder (T5, for text-to-text).
8. **Why they won:** parallelism enabled training at unprecedented scale. Scale wins.

Home Assignment

Task: Explore attention patterns in a pre-trained model on political text.

Instructions:

1. Use the practical code as a starting point.
2. Pick **three** sentences from UK parliamentary speeches (or your own political corpus). Each should have an interesting feature: e.g., negation, a pronoun, a coordinated structure, or a long-range dependency.
3. For each sentence, identify **at least one** attention head whose pattern you can interpret. Save the heatmap.
4. For each interpretable head, write 2–3 sentences explaining what relationship it seems to encode and how you tested this.
5. Compare attention patterns in an **early layer** (e.g., layer 2) vs. a **late layer** (e.g., layer 10) for the same sentence. What changes?
6. Write a **2-page report** with your findings.

Submit: Python script (or Colab notebook) + 2-page report + saved heatmaps. Due before Lecture 7.

Readings

Required:

- Jurafsky, D. & Martin, J.H. (2025). *Speech and Language Processing*, Chapter 8: Transformers. https://web.stanford.edu/~jurafsky/slp3/8.pdf

Recommended:

- Vaswani, A., Shazeer, N., Parmar, N., et al. (2017). "Attention Is All You Need." *NeurIPS 2017*. arXiv:1706.03762.
- Alammar, J. (2018). "The Illustrated Transformer." http://jalammar.github.io/illustrated-transformer/ (Intuitive visual explanation of the Transformer)

Next lecture: Transfer Learning & Fine-Tuning BERT

Putting the Transformer to work on a political classification task — the culmination of the course