NATURAL LANGUAGE PROCESSING

Attention mechanisms & Transformer architecture

Modern NLP Foundations & Chatbot development

Learning objectives

1. Understand the motivation and mechanics of Attention mechanism in Seq2Seq models
2. Master Self-Attention with Query, Key, Value framework and Scaled Dot-Product
3. Explain Multi-Head Attention and why multiple heads improve performance
4. Understand Transformer architecture: Encoder, Decoder, Positional Encoding
5. Build Chatbots using attention-based models and modern LLM architectures
6. Evaluate and deploy conversational AI systems with proper metrics

Session agenda

Part 1

Attention mechanisms: a review

Seq2Seq recap, attention core idea, score functions

Part 2

Self-Attention & Multi-head

Q, K, V framework, scaled dot-product, multiple heads

Part 3

Transformer architecture

Positional encoding, encoder-decoder, BERT/GPT/T5

Part 4

Chatbot development

NLU pipeline, intent & slots, dialogue state tracking

Part 5

Implementation & Evaluation

HuggingFace, metrics, common pitfalls

Part 6

Lab & Final Assignment

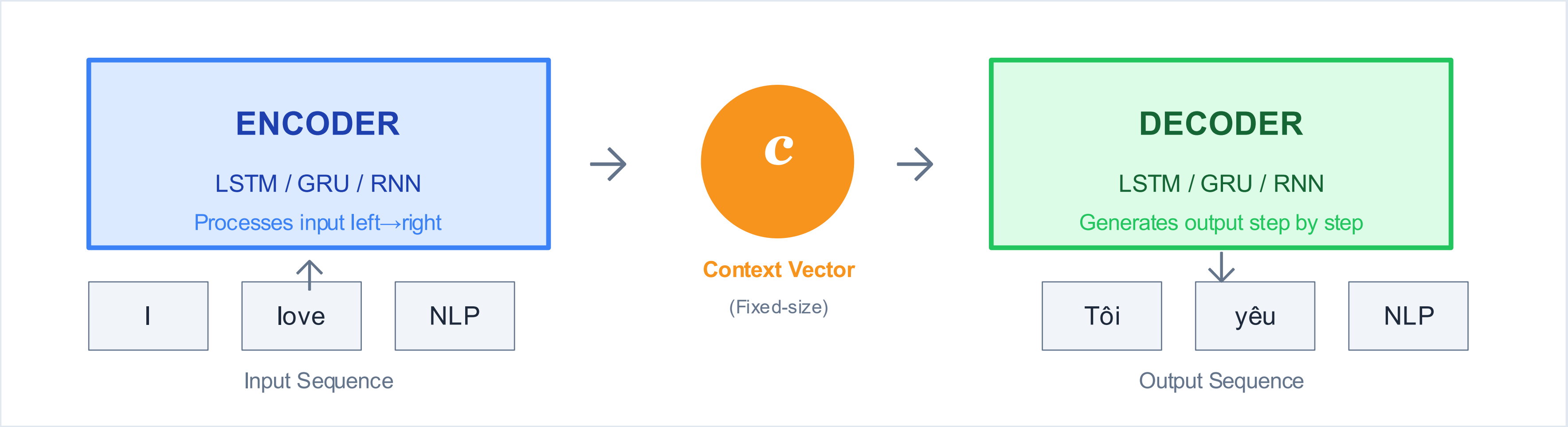
Hands-on exercises, project requirements

Attention mechanisms: A review

From Seq2Seq bottleneck to dynamic context

- Seq2Seq Architecture Recap
- The Bottleneck Problem
- Attention Score Functions
- Context Vector Computation

Recap: Seq2Seq (Encoder-Decoder) architecture



Key Equations

Encoder: $h_t = f(x_t, h_{t-1})$

Context: $c = h_t$(final hidden state)

Decoder: $s_t = g(y_{t-1}, s_{t-1}, c)$

Applications

Machine Translation (EN→VN)

Text Summarization

Chatbots & Dialogue Systems

Question Answering

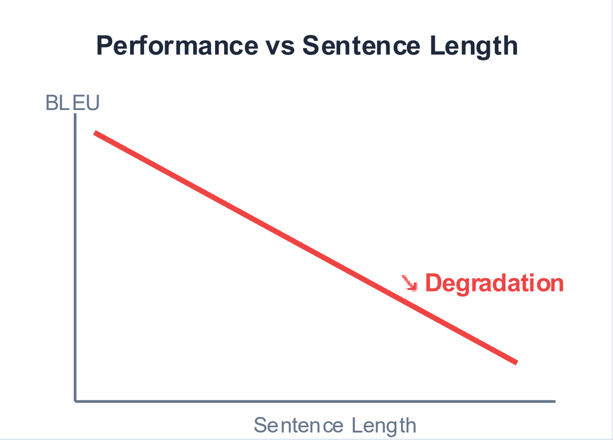
The Bottleneck problem in Seq2Seq

The Problem

The entire input sequence is compressed into a single fixed-size vector c.

For long sequences, this causes severe information loss!

Long input sequence (50+ tokens)
→
c


Performance vs Sentence Length

Key Insight

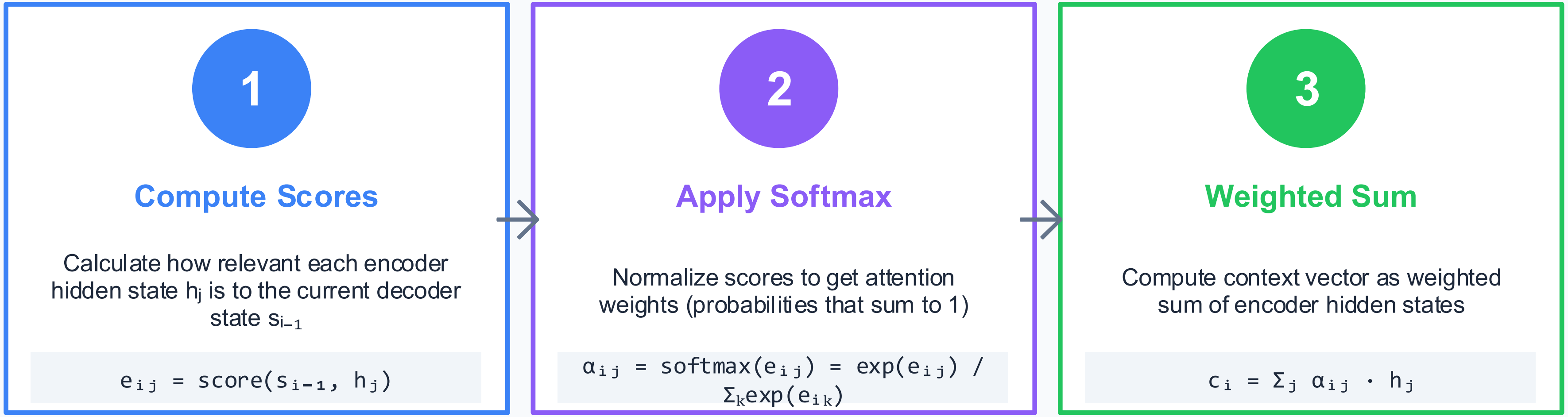
When translating a word, we don't need ALL input information — just the relevant parts! This observation leads to the Attention mechanism.

Solution: Attention Mechanism

Instead of using one fixed context vector, dynamically focus on different parts of the input at each decoding step.

Attention: The core idea

At each decoding step, compute a weighted combination of ALL encoder hidden states based on their relevance to the current output.



1

Compute Scores

Calculate how relevant each encoder hidden state h_j is to the current decoder state s_{i-1}

$e_{ij} = \text{score}(s_{i-1}, h_j)$

2

Apply Softmax

Normalize scores to get attention weights (probabilities that sum to 1)

$\alpha_{ij} = \text{softmax}(e_{ij}) = \exp(e_{ij}) / \Sigma_k \exp(e_{ik})$

3

Weighted Sum

Compute context vector as weighted sum of encoder hidden states

$c_i = \Sigma_j \alpha_{ij} \cdot h_j$

Human Intuition

I love machine learning → Focus on relevant words → Tôi yêu học máy

Attention score functions

Different ways to compute relevance score between decoder state s_{i-1} and encoder hidden state h_j:

Dot Product

(Luong, 2015)

$$\text{score}(s,\ h) = s^{T}h$$

✓ **Pros:**

Simple, fast, no parameters

✗ **Cons:**

Requires same dimensions

General (Bilinear)

(Luong, 2015)

$$\text{score}(s,\ h) = s^{T}W_{a}h$$

✓ **Pros:**

Learnable, flexible

✗ **Cons:**

Needs d×d parameter matrix

Concat (Additive)

(Bahdanau, 2014)

$$\text{score} = v_{a}^{T}\tanh(W_{a}[s;h])$$

✓ **Pros:**

Most expressive

✗ **Cons:**

Slower, more parameters

Transformer uses Scaled Dot-Product: divides by $\sqrt{d_k}$ to prevent softmax saturation

Attention weights: step-by-step

$$\alpha_{ij} = \text{softmax}(e_{ij}) = \exp(e_{ij}) / \Sigma_k \exp(e_{ik})$$

Example: Translating "I love machine learning" → "Tôi ..."

Step 1: Compute raw scores (dot product)

$e_{11} = s_0 \cdot h_1 = 2.1$ ("I")

$e_{12} = s_0 \cdot h_2 = 0.5$ ("love")

$e_{13} = s_0 \cdot h_3 = 0.3$ ("machine")

$e_{14} = s_0 \cdot h_4 = 0.1$ ("learning")

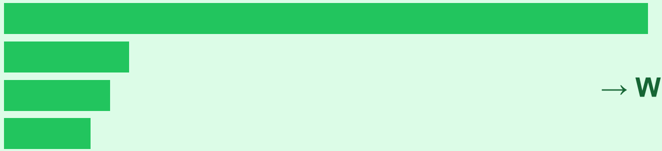
Step 2: Apply softmax

exp(2.1)=8.17, exp(0.5)=1.65
exp(0.3)=1.35, exp(0.1)=1.11

Sum = 12.28

Step 3: Attention weights (normalized probabilities)

I	$\alpha = 0.67$
love	$\alpha = 0.13$
machine	$\alpha = 0.11$
learning	$\alpha = 0.09$



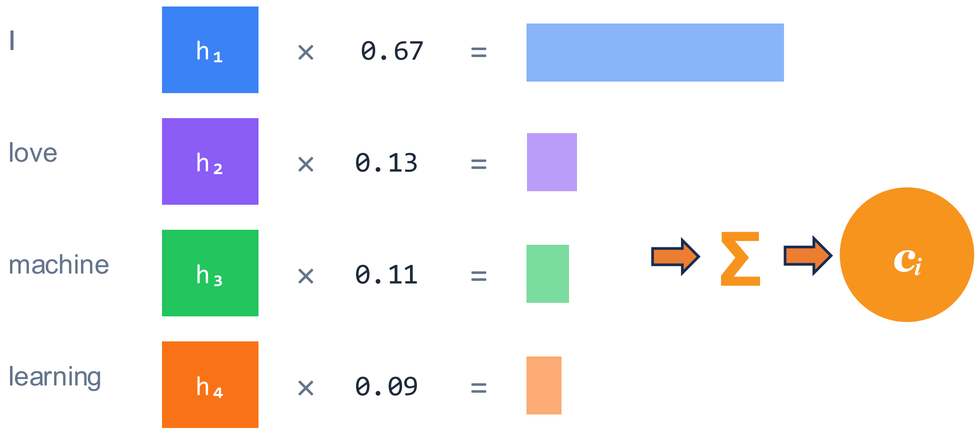
→ **When generating "Tôi", model focuses 67% on "I"**

Key Properties: $\alpha_{ij} \in [0,1]$ • $\Sigma_j \alpha_{ij} = 1$
- Higher weight = more relevance
- Differentiable!

Context vector computation

$$c_i = \Sigma_j \; \alpha_{ij} \cdot h_j \quad \text{(Weighted sum of encoder hidden states)}$$

Visual: How context vector is computed



Numerical Example

If each h_j is a 3-dim vector:

$h_1 = [0.2, 0.8, 0.1]$
$h_2 = [0.5, 0.3, 0.2]$
$h_3 = [0.1, 0.4, 0.9]$
$h_4 = [0.3, 0.1, 0.7]$

$c = 0.67 \times h_1 + 0.13 \times h_2$
$\quad + 0.11 \times h_3 + 0.09 \times h_4$

$c = [0.24, 0.62, 0.24]$

→ **Dominated by h_1 ("I")**

Key: Context vector c_i is DIFFERENT for each decoder step — dynamically adapts to what's being generated!

Attention visualization: alignment matrix

Attention Alignment Matrix (EN → VN)

	I	love	machine	learning
Tôi	0.90	0.05	0.03	0.02
yêu	0.05	0.85	0.05	0.05
học	0.02	0.08	0.45	0.45
máy	0.02	0.03	0.50	0.45

Interpretation

"Tôi" → "I" (0.90)
"yêu" → "love" (0.85)
"học máy" → "machine" + "learning"
(multi-word alignment!)

Benefits

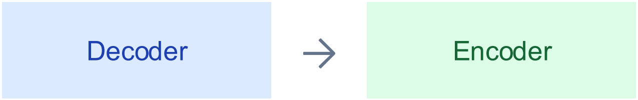
- Interpretable alignments
- Debug translation errors
- Understand model behavior
- Detect attention issues

Multi-word alignment: "machine learning" → "học máy" — attention handles this naturally!

Cross-attention vs Self-attention

Cross-Attention

Used in: Seq2Seq, Encoder-Decoder



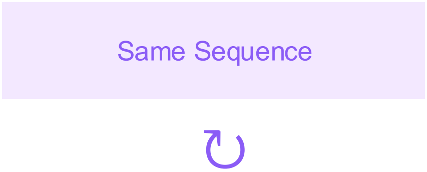
Key points:

- Query: from decoder (target)
- Key, Value: from encoder (source)
- Connects TWO sequences
- Used for translation alignment

$$\mathtt{score}(\mathtt{decoder}_{\mathtt{state}},\ \mathtt{encoder}_{\mathtt{state}})$$

Self-Attention

Used in: Transformers, BERT, GPT



Key points:

- Q, K, V all from SAME sequence
- Each position attends to all others
- Captures internal relationships
- Foundation of Transformers

$$\mathtt{score}(\mathtt{position}_{i},\ \mathtt{position}_{j})$$

Key: Cross = between sequences | Self = within same sequence. Transformer uses BOTH!

Why Self-attention for NLP?

RNN/LSTM Limitations

- Sequential processing: O(n) steps
- Long-range dependencies hard
- Cannot parallelize training
- Vanishing gradients for long seq

Self-Attention Advantages

- Parallel processing: O(1) steps
- Direct connection to any position
- Fully parallelizable on GPU
- Constant path length

Example: "The cat sat on the mat because it was tired"

Self-attention directly connects "it" → "cat" in 1 step. RNN needs 5 steps to propagate this information.

Summary: Attention Mechanisms

1 **Bottleneck Problem**

Fixed context vector loses info for long sequences

2 **Attention Solution**

Dynamically focus on relevant input parts per step

3 **Score Functions**

Dot Product, General, Concat to compute relevance

4 **Attention Weights**

Softmax gives probability distribution over inputs

5 **Self-Attention**

Each position attends to all positions in same sequence

Essential Formulas

Score Functions:

Dot: $s^T h$

General: $s^T W_a h$

Concat: $v_a^T \tanh(W_a[s;h])$

Attention Weight:

$\alpha_{ij} = \text{softmax}(e_{ij})$

Context Vector:

$c_i = \Sigma_j \alpha_{ij} \cdot h_j$

Self-attention & Multi-head attention

The foundation of transformer architecture

- Query, Key, Value Framework
- Scaled Dot-Product Attention
- Why Scale by $\sqrt{d_k}$?
- Multi-Head Attention
- Parameter Analysis & Dimensions

Introducing Query, Key, Value (Q, K, V)

Self-Attention uses three learned projections of the input: Query (Q), Key (K), and Value (V)

Q

Query

"What am I looking for?"

Represents the current position's question to ask other positions

K

Key

"What do I contain?"

Represents information each position advertises about itself

V

Value

"What do I provide?"

The actual content to be retrieved when a query matches a key

Attention Flow: Q matches K → score → softmax → retrieve V → **Output**

Database analogy: understanding Q, K, V

Think of attention as a "soft" database lookup — instead of exact matching, we use similarity scores

Traditional Database Lookup

`Query: SELECT value WHERE key = 'cat'`

Key	Value
dog	[0.2, 0.8, 0.1]
cat	[0.9, 0.3, 0.5]
bird	[0.1, 0.6, 0.7]

Result: Exact match → [0.9, 0.3, 0.5]

Binary: either 0 or 1 (match/no match)

Attention (soft lookup)

`Query: "kitten" (similar to cat)`

Key	Score	Value
dog	**0.2**	[0.2, 0.8, 0.1]
cat	**0.7**	[0.9, 0.3, 0.5]
bird	**0.1**	[0.1, 0.6, 0.7]

Result: Weighted sum of all values

$= 0.2\times v_1 + 0.7\times v_2 + 0.1\times v_3$

Continuous: soft weights $\in [0,1]$

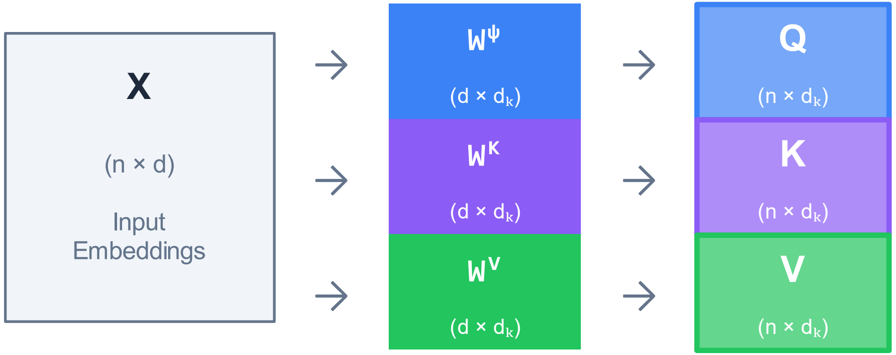
Attention is differentiable — we can learn Q, K, V through backpropagation!

Creating Q, K, V: linear projections

$$Q = X \cdot W^{\psi} \qquad K = X \cdot W^{K} \qquad V = X \cdot W^{V}$$

Where X is the input sequence, and W^{φ}, W^{K}, W^{V} are learnable weight matrices

Visual: How Q, K, V are created from input X



Dimensions

n = sequence length
d = input embedding dimension
d_k = key/query dimension
d_v = value dimension

Typically: $d_k = d_v = d/h$
(h = number of heads)

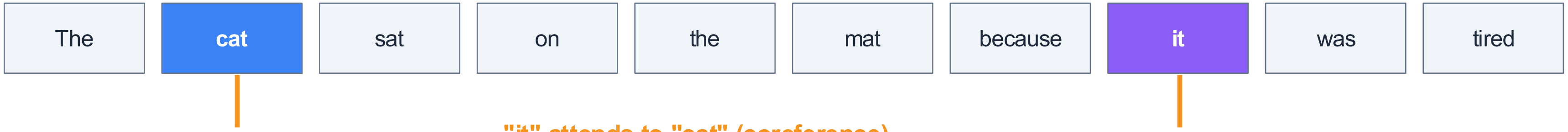
Same input X → Different projections

W matrices are LEARNED during training

Self-Attention: each position attends to all

In self-attention, Q, K, V all come from the SAME sequence — each position can attend to every other position

Example: "The cat sat on the mat because it was tired"



"it" attends to "cat" (coreference)

Self-Attention Matrix (simplified):

	cat	sat	it	was
cat	0.9	0.1	0.0	0.0
sat	0.3	0.5	0.1	0.1
it	0.7	0.1	0.1	0.1
was	0.2	0.2	0.3	0.4

← "it" strongly attends to "cat" (0.7)

Why Self-Attention Works

- Captures long-range dependencies
- Resolves coreference (it → cat)
- Parallel computation (no RNN!)
- Learns multiple relationships

Scaled dot-product attention

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

The core formula of Transformer attention — memorize this!

Formula Breakdown

Q	Query matrix (n × d_k) — what each position is looking for	QK^T	Attention scores matrix (n × n) — similarity between all pairs
K	Key matrix (n × d_k) — what each position contains	$\sqrt{d_k}$	Scaling factor — prevents softmax saturation (explained next)
V	Value matrix (n × d_v) — actual content to retrieve	softmax	Normalizes scores to probabilities (rows sum to 1)

Output Dimensions: (n × n) × (n × d_v) = (n × d_v) — same shape as V, but with attended information

Why Scale by $\sqrt{d_k}$? preventing gradient vanishing

Problem Without Scaling

For high d_k (e.g., 64), the dot product QK^T grows very large:

If q, k ~ N(0,1), then:
$q \cdot k$ has variance = d_k

For d_k = 64:
Variance = 64, Std = 8

Large values → Softmax saturates → Gradients vanish!

Solution: Scale by $\sqrt{d_k}$

Dividing by $\sqrt{d_k}$ normalizes the variance:

$\mathrm{Var}(q \cdot k \,/\, \sqrt{d_k}) = d_k \,/\, d_k = 1$

For d_k = 64:
$\sqrt{64}$ = 8
New Variance = 1, Std = 1

Stable softmax → Healthy gradients → Better training!

Softmax Behavior Comparison

Without scaling (scores = [8, 4, 0, -4]):

0.982	0.018	0.000	0.000

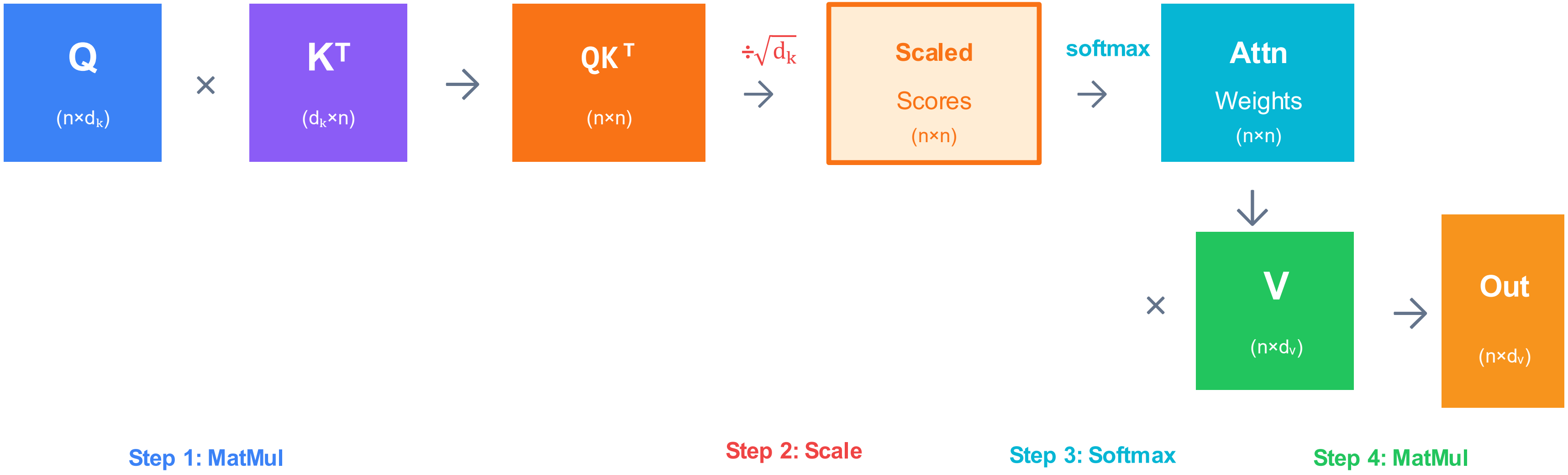
→ Almost one-hot! Gradients ≈ 0

With scaling (scores = [1, 0.5, 0, -0.5]):

0.39	0.24	0.14	0.09

→ Smooth distribution, healthy gradients!

Scaled dot-product: computation flow



Example

Input: "I love NLP" with $d_k = 4$

Q (3×4)

```
I:    [1, 0, 1, 0]
love: [0, 1, 1, 0]
NLP:  [1, 1, 0, 1]
```

K (3×4)

```
I:    [1, 1, 0, 0]
love: [0, 1, 1, 1]
NLP:  [1, 0, 1, 1]
```

V (3×4)

```
I:    [0.1, 0.2, 0.3, 0.4]
love: [0.5, 0.6, 0.7, 0.8]
NLP:  [0.9, 1.0, 1.1, 1.2]
```

Step 1: QK^T (raw scores)

```
           I     love    NLP
I:      [ 1,     1,      2 ]
love:   [ 1,     2,      1 ]
NLP:    [ 1,     2,      2 ]
```

Step 2: ÷ $\sqrt{4}$ = ÷ 2

```
           I     love    NLP
I:      [0.5,  0.5,    1.0]
love:   [0.5,  1.0,    0.5]
NLP:    [0.5,  1.0,    1.0]
```

Step 3: Softmax (each row)

```
           I     love    NLP
I:      [0.27, 0.27,   0.45]
love:   [0.24, 0.40,   0.24]
NLP:    [0.21, 0.35,   0.35]
```

Step 4: Attn × V = Output

```
I:     [0.57, 0.66, 0.75, 0.84]
love:  [0.48, 0.56, 0.64, 0.72]
NLP:   [0.54, 0.62, 0.70, 0.78]
```

Weighted combination of V rows!

Multi-head attention: Why multiple heads?

Single Head Limitation

One attention head can only learn ONE type of relationship:

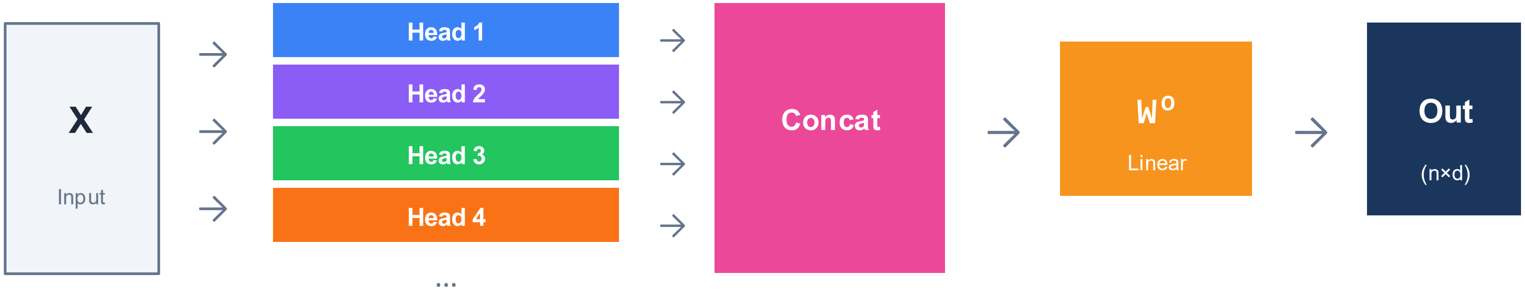
- Maybe it learns syntax
- OR maybe semantics
- OR maybe position
- But NOT all at once!

Multi-Head Solution

Run multiple attention heads in parallel, each learning different relationships:

- Head 1: syntactic structure
- Head 2: coreference
- Head 3: semantic similarity
- Concatenate all insights!

Multi-Head Architecture



Different heads learn different aspects: syntax, semantics, coreference, position — then combine them!

Multi-Head Attention: the formula

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

$$\text{where } \mathrm{head}_i = \mathrm{Attention}(Q \cdot W_i^{\psi}, K \cdot W_i^K, V \cdot W_i^V)$$

h = number of heads (typically 8 or 12)

Understanding Each Component

W_i^{ψ}, W_i^K, W_i^V	Projection matrices for head i	$(d \times d_k)$, $(d \times d_k)$, $(d \times d_v)$
head_i	Output of attention head i	$(n \times d_v)$
Concat(...)	Concatenate all head outputs	$(n \times h \cdot d_v)$
W^O	Output projection matrix	$(h \cdot d_v \times d)$

Insight

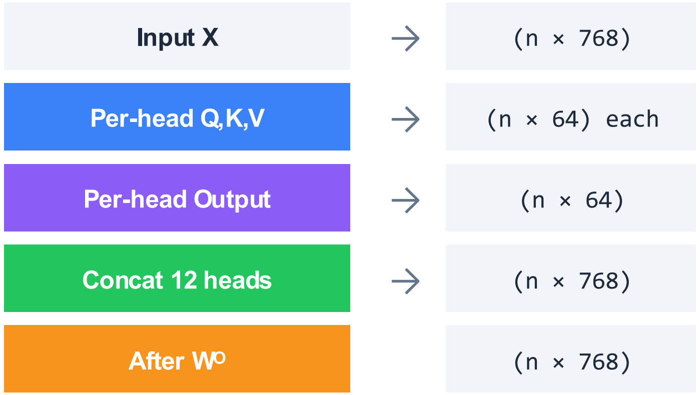
Each head has its own W^{φ}, W^K, W^v matrices, allowing it to learn different patterns!

BERT-base: $d=768$, $h=12$, $d_k=d_v=64$ | GPT-2: $d=768$, $h=12$, $d_k=d_v=64$ | GPT-3: $d=12288$, $h=96$, $d_k=d_v=128$

Parameter & dimension analysis

Example: BERT-base configuration — d = 768, h = 12 heads, $d_k = d_v = 64$

Dimension Flow



Input X	→	(n × 768)
Per-head Q,K,V	→	(n × 64) each
Per-head Output	→	(n × 64)
Concat 12 heads	→	(n × 768)
After W^O		(n × 768)

Why $d_k = d/h = 768/12 = 64$?

Concat restores original dim!

Parameter Count

Per head (W^{φ}, W^K, W^V):

3 × (768 × 64) = 147,456

All 12 heads:

12 × 147,456 = 1,769,472

Output W^O:

768 × 768 = 589,824

Total: ~2.36M params

Model Comparison

Model	d	h	$d_k = d/h$	MHA Params
BERT-base	768	12	64	2.36M
GPT-2	768	12	64	2.36M
GPT-3 (175B)	12288	96	128	603M

What different attention heads learn

Research has shown that different heads specialize in different linguistic patterns (Clark et al., 2019)

Syntactic Heads

Attend to grammatical structure

e.g., subject→verb, noun→adjective

Positional Heads

Attend to nearby positions

e.g., previous token, next token

Coreference Heads

Link pronouns to entities

e.g., "it" → "the cat"

Semantic Heads

Attend to related meanings

e.g., "bank" → "money", "river"

Example: "The cat sat on the mat because it was tired"

Head 3 (Coreference): "it" strongly attends to "cat" (0.72) | Head 7 (Positional): "sat" attends to "cat" (0.45) | Head 11 (Syntactic): "tired" attends to "was" (0.68)

Summary: self-attention & multi-Head

1 **Q, K, V Framework**

Query asks, Key advertises, Value provides content

2 **Scaled Dot-Product**

Attention = softmax($QK^T/\sqrt{d_k}$) · V

3 **Why $\sqrt{d_k}$ Scaling**

Prevents gradient vanishing from softmax saturation

4 **Multi-Head Attention**

Multiple heads learn different relationship types

5 **Dimension Analysis**

$d_k = d/h$ ensures output matches input dimension

Formulas

Linear Projections:

$Q = XW^{\psi}, K = XW^K, V = XW^V$

Scaled Dot-Product:

$\mathrm{Attn} = \mathrm{softmax}(QK^T/\sqrt{d_k})V$

Multi-Head:

$\mathrm{MHA} = \mathrm{Concat}(\mathrm{heads})W^O$

Typical Config:

$d=768, h=12, d_k=64$

Transformer architecture

"Attention Is All You Need" (Vaswani et al., 2017)

The architecture that revolutionized NLP and AI

- Why Transformers? RNN vs Self-Attention
- Positional Encoding — Adding Position Information
- Encoder Block — Self-Attention + FFN
- Decoder Block — Masked Attention + Cross-Attention
- BERT vs GPT vs T5 — Architectural Variants

Why transformers? From RNN to self-attention

RNN/LSTM Problems

Sequential Processing
Must process tokens one-by-one, O(n) steps

Long-Range Dependencies
Information decays over distance

No Parallelization
Cannot utilize GPU parallelism

Vanishing Gradients
Hard to train on long sequences

Transformer Solutions

Parallel Processing
All positions computed simultaneously

Direct Connections
Any token can attend to any other in O(1)

Full GPU Utilization
Matrix operations are highly parallel

Stable Gradients
Residual connections + LayerNorm

Key Innovation: Replace recurrence entirely with Self-Attention — "Attention Is All You Need"

Training Speed: Transformer trains ~10x faster than RNN on same data (due to parallelization)

Original paper: Achieved SOTA on WMT EN→DE translation in 3.5 days on 8 GPUs (vs weeks for RNN)

Positional Encoding: Why is it needed?

The Problem: Self-Attention is Position-Agnostic!

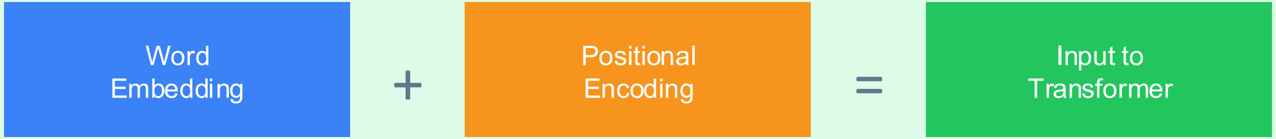
Self-attention treats input as a SET, not a SEQUENCE. Without position info, "dog bites man" = "man bites dog"!

Example: Position matters for meaning!

Sentence A:	The	dog	bites	the	man	→ Dog is the actor
Sentence B:	The	man	bites	the	dog	→ Man is the actor

Same words, different order → Completely different meaning!

Solution: Add Positional Encoding to Input Embeddings



Word Embedding + Positional Encoding = Input to Transformer

PE is added, not concatenated
(keeps dimension d unchanged)

Positional Encoding: sinusoidal formulas

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

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

pos = position in sequence (0, 1, 2, ...), i = dimension index (0 to d/2-1), d = model dimension

Why sinusoidal functions?

1. Unique encoding per position
Each position has distinct pattern

2. Bounded values [-1, 1]
Won't dominate embeddings

3. Relative positions learnable
PE(pos+k) is linear function of PE(pos)

4. Generalizes to longer sequences
Can extrapolate beyond training length

different wavelengths

Dimension 0-1 (i=0):

$\text{wavelength} = 2\pi \approx 6.28$

→ Changes rapidly with position

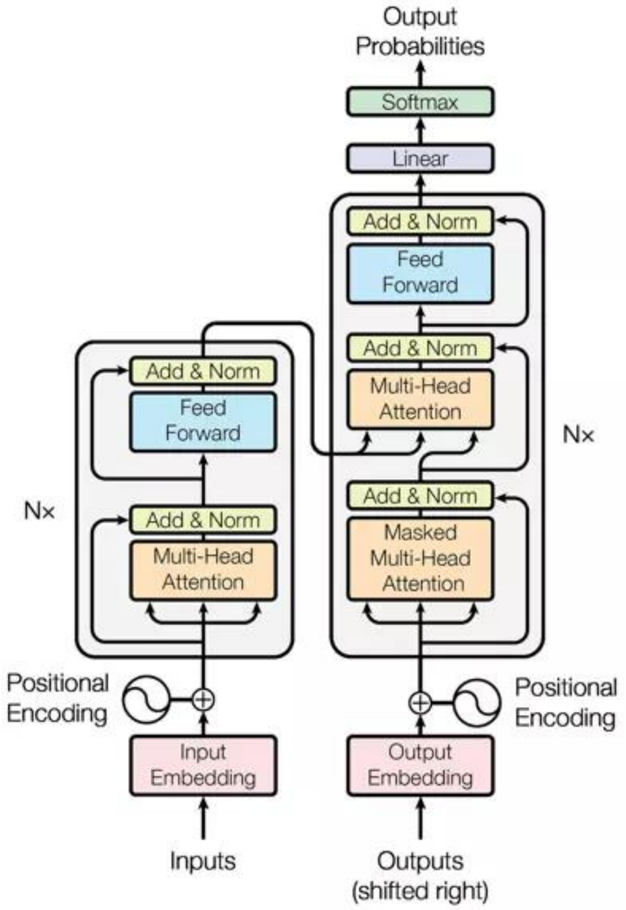
Dimension d-2, d-1 (i=d/2-1):

$\text{wavelength} = 2\pi \times 10000 \approx 62{,}832$

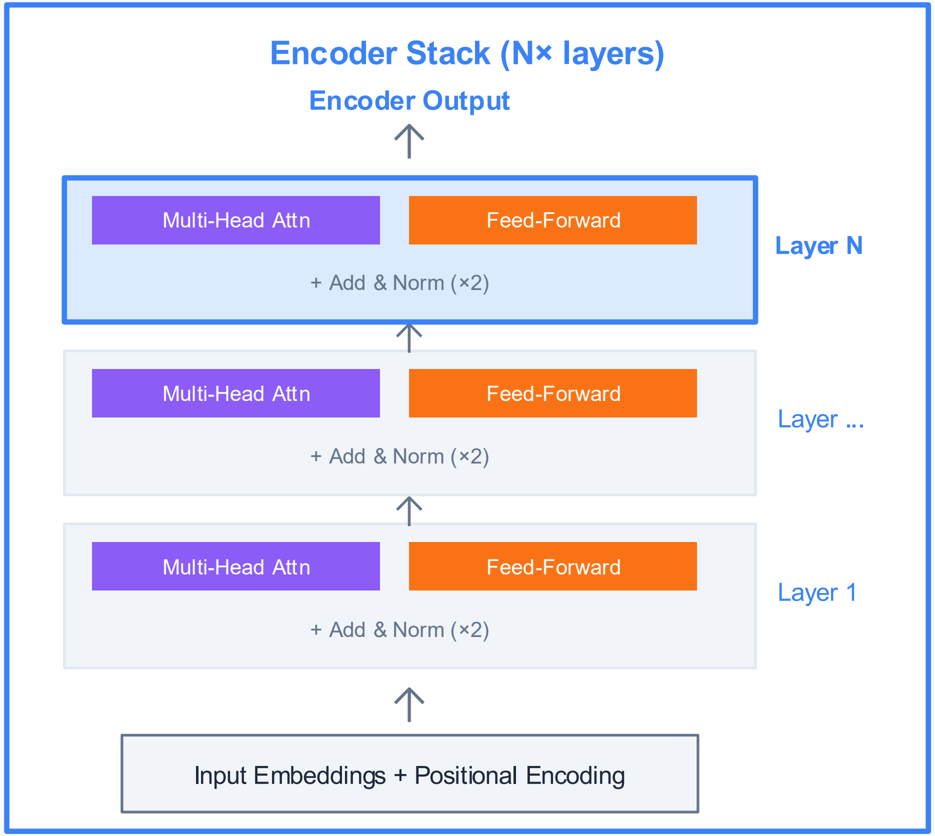
→ Changes very slowly

This creates a spectrum from fine-grained to coarse-grained position information!

Transformer architecture



Transformer Encoder: Overview



Each Encoder Layer Contains:

1. Multi-Head Self-Attention

Each position attends to all positions in the input

2. Add & Normalize

Residual connection + Layer Normalization

3. Feed-Forward Network

Two linear layers with ReLU activation

4. Add & Normalize

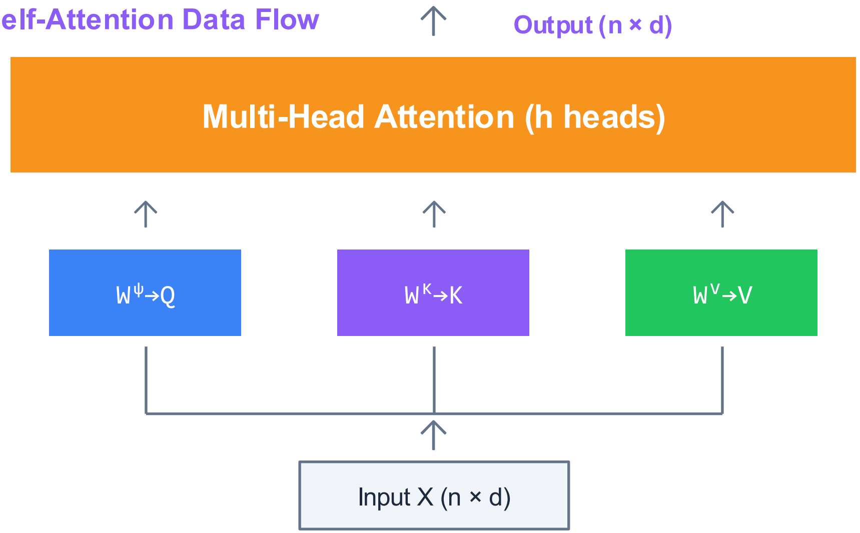
Another residual + LayerNorm

BERT-base: N=12, GPT-2: N=12, GPT-3: N=96

Encoder: Self-Attention Sub-Layer

```
SelfAttn(X) = MultiHead(Q=X, K=X, V=X) — All from same input!
```

Self-Attention Data Flow



Key Points

Bidirectional Attention
Each position can see ALL other positions (past and future)

No Masking in Encoder
Unlike decoder, encoder sees full sequence

Parallel Computation
All n positions computed simultaneously

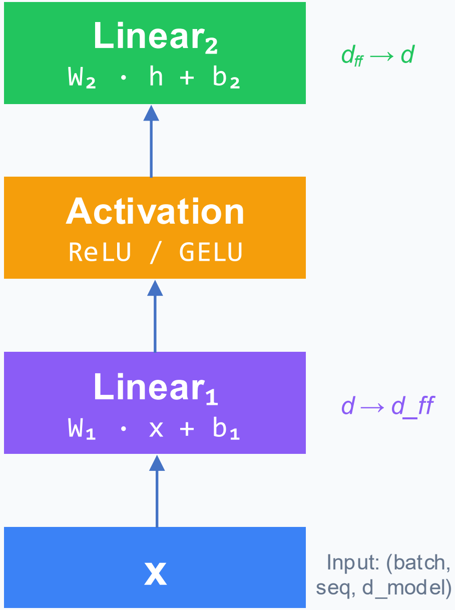
Context Mixing
Each output is a mix of all inputs

Output shape is same as input (n × d) — this allows stacking multiple encoder layers!

Encoder: Feed-Forward Network (FFN)

$$\mathrm{FFN}(x) = \max(0,\ x \cdot W_1 + b_1) \cdot W_2 + b_2$$

$= \mathrm{ReLU}(x \cdot W_1 + b_1) \cdot W_2 + b_2$ (Two linear layers with ReLU in between)



Key Properties

Expansion then Compression
$d \to d_{ff}$ (expand) $\to d$ (compress)
Typically $d_{ff} = 4d$ (e.g., 768→3072)

Position-wise
Same FFN applied to each position
No interaction between positions

Non-linearity
ReLU adds non-linear transformation
(GELU used in BERT/GPT)

Parameters: W_1 ($d \times d_{ff}$) + W_2 ($d_{ff} \times d$) + biases ≈ $2 \times d \times d_{ff}$

BERT-base: 768×3072×2 ≈ 4.7M params per layer × 12 layers = 56M params (just for FFN!)

Activation functions in FFN

ReLU (Original Transformer)

$\mathrm{ReLU}(x) = \max(0, x)$

Simple, fast, but "dead neurons" issue

Used in: Original Transformer (2017)

GELU (Modern)

$\mathrm{GELU}(x) = x \cdot P(X \leq x) = x \cdot \Phi(x)$

$$= 0.5x(1 + \tanh\left(\sqrt{\frac{2}{\pi}}(x + 0.044715x^3\right))$$

Used in: BERT, GPT, most modern LLMs

SwiGLU (Modern LLMs: LLaMA, Mistral, Qwen)

$\mathrm{SwiGLU}(x,W,V,b,c) = \mathrm{Swish}(xW+b) \odot (xV+c)$

where $\mathrm{Swish}(x) = x \cdot \sigma(x)$

Evolution: ReLU (2017) → GELU (2018) → SwiGLU (2020+)

Layer Normalization & Residual Connections

Layer Normalization

$$LN(x) = \gamma \cdot (x - \mu) / \sigma + \beta$$

μ = mean(x), σ = std(x), γ and β are learned

Why Layer Norm (not Batch Norm)?

- Normalize across features (d dimension)
- Independent of batch size
- Works with variable sequence lengths
- More stable for sequence models

Residual Connections

```
Output = x + SubLayer(x)
```

Why Residual Connections?

- Gradient flows directly (skip connection)
- Enables training very deep networks
- Model can learn "do nothing" easily
- Proven in ResNet (He et al., 2015)

Combined: Add & Norm (Post-LN Transformer)

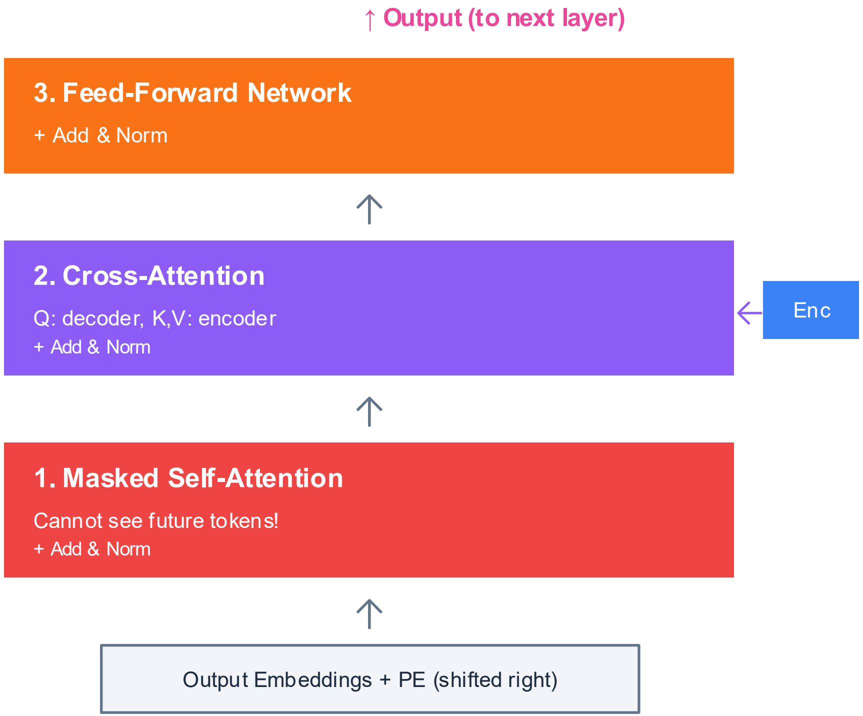
```
output = LayerNorm(x + SubLayer(x))
```

Applied twice in each encoder layer: once after Self-Attention, once after FFN

Pre-LN: LN(x) + SubLayer(LN(x)) — more stable for deep models

Transformer Decoder: Overview

Decoder Layer Structure



↑ Output (to next layer)

3. Feed-Forward Network

+ Add & Norm

2. Cross-Attention

Q: decoder, K,V: encoder

+ Add & Norm

Enc

1. Masked Self-Attention

Cannot see future tokens!

+ Add & Norm

Output Embeddings + PE (shifted right)

Decoder has 3 Sub-layers

1 **Masked Self-Attention**

Attend to previous output tokens only (causal)

2 **Cross-Attention**

Attend to encoder output (source sequence)

3 **Feed-Forward Network**

Same as encoder FFN (position-wise)

Each sub-layer has Add & Norm:
output = LayerNorm(x + Sublayer(x))

Decoder: Masked Self-Attention

Problem: At inference, decoder generates tokens one-by-one — it cannot see future tokens!

Causal Mask (Lower Triangular)

	<s>	I	love	NLP
<s>	1	-∞	-∞	-∞
I	1	1	-∞	-∞
love	1	1	1	-∞
NLP	1	1	1	1

🟩 = can attend 🟥 = masked (−∞ → softmax=0)

Masked Attention Formula

$$\text{Attn} = \text{softmax}((QK^T + M) / \sqrt{d_k}) \cdot V$$

where $M[i,j] = 0$ if $j \le i$, else $-\infty$

softmax(−∞) = 0, so future tokens get zero weight!

Generation Example

Step 1: Generate "I"
→ sees only <start>

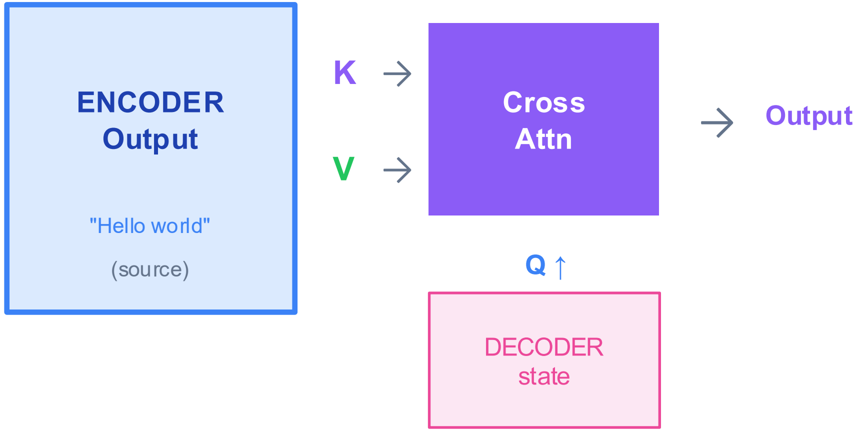
Step 2: Generate "love"
→ sees <start>, "I"

Step 3: Generate "NLP"
→ sees <start>, "I", "love"

Decoder: Cross-Attention

$$\mathrm{CrossAttn}(Q_{dec}, K_{enc}, V_{enc}) = \mathrm{softmax}(Q_{dec} \cdot K_{enc}^{T} / \sqrt{d_k}) \cdot V_{enc}$$

Cross-Attention Data Flow



How Cross-Attention Works

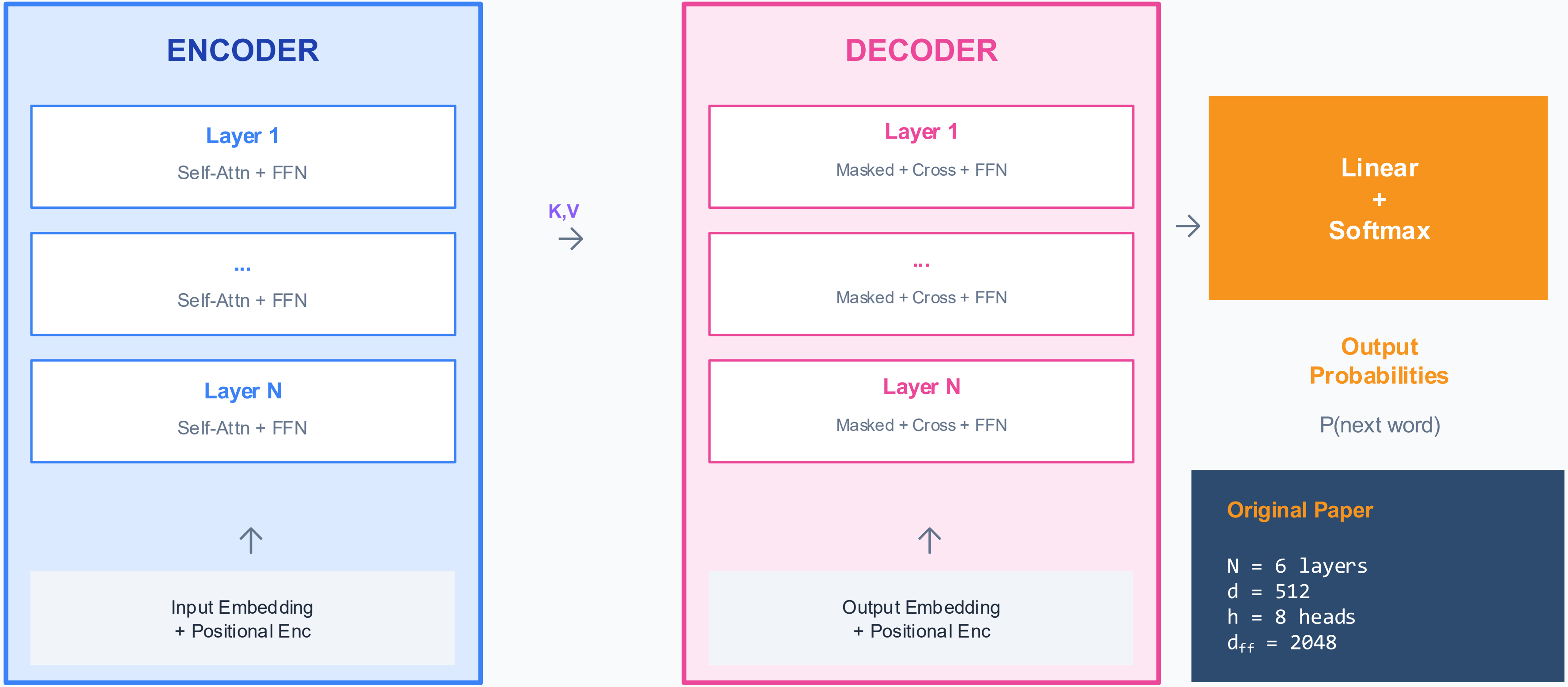
Q comes from Decoder
"What info do I need to generate the next target word?"

K, V come from Encoder
"Here's the source sentence context for you to use."

Result
Decoder focuses on relevant source words when generating each target word

Translation: "Hello world" → "Xin chào thế giới" — when generating "chào", decoder attends strongly to "Hello"

Transformer Architecture



ENCODER

Layer 1
Self-Attn + FFN

...
Self-Attn + FFN

Layer N
Self-Attn + FFN

Input Embedding
+ Positional Enc

K,V →

DECODER

Layer 1
Masked + Cross + FFN

...
Masked + Cross + FFN

Layer N
Masked + Cross + FFN

Output Embedding
+ Positional Enc

→

Linear
+
Softmax

Output Probabilities

P(next word)

Original Paper

N = 6 layers
d = 512
h = 8 heads
d_{ff} = 2048

Encoder: bidirectional self-attention | Decoder: masked self-attention + cross-attention to encoder

Transformer variants: BERT vs GPT vs T5

BERT

(2018, Google)

Bidirectional Encoder

Architecture

Encoder-only

Attention

Bidirectional

Pre-training

MLM + NSP

Best for

Classification, NER, QA

Encoder

GPT

(2018-2023, OpenAI)

Generative Pre-trained

Architecture

Decoder-only

Attention

Causal (left→right)

Pre-training

Next token prediction

Best for

Text generation, Chat

Decoder

T5

(2019, Google)

Text-to-Text Transfer

Architecture

Encoder-Decoder

Attention

Enc: bi, Dec: causal

Pre-training

Span corruption

Best for

Translation, Summary

Enc → Dec

Modern trend: Decoder-only (GPT-style) dominates for generation; Encoder-only (BERT) for understanding tasks

Training Transformers: key considerations

Training Setup

Optimizer: AdamW

β_1=0.9, β_2=0.98, weight decay=0.01

Learning Rate Schedule:

Warmup + Linear/Cosine decay

Regularization:

Dropout (0.1)

Model Size Comparison

Model	Params	Layers	d_model	Heads	Training Data
BERT-base	110M	12	768	12	16GB
GPT-2	1.5B	48	1600	25	40GB
GPT-3	175B	96	12288	96	570GB
LLaMA 2	70B	80	8192	64	2T tokens
GPT-4	~1.8T*	?	?	?	~13T tokens*

* Estimated/rumored

Summary: transformer architecture

1 **Positional Encoding**

Sinusoidal functions add position info to embeddings

2 **Encoder Block**

Self-Attention + FFN + Add&Norm (bidirectional)

3 **Decoder Block**

Masked Self-Attn + Cross-Attn + FFN (causal)

4 **Residual + LayerNorm**

Enable deep networks with stable gradients

5 **BERT/GPT/T5**

Encoder-only / Decoder-only / Encoder-Decoder variants

Formulas

Positional Encoding:

$PE(pos,2i) = sin(pos/10000^{(2i/d)})$

FFN:

$FFN(x) = ReLU(xW_1+b_1)W_2+b_2$

Add & Norm:

$LayerNorm(x + Sublayer(x))$

Masked Attention:

$M[i,j] = 0 \text{ if } j \le i \text{ else } -\infty$

Chatbot development

From rule-based to LLM-powered Conversational AI

- Chatbot Types & Architecture
- NLU Pipeline: Intent & Slot Filling
- Dialogue State Tracking
- LLM-based Chatbots & RAG
- Implementation with HuggingFace

What is a Chatbot?

A chatbot is a software application that simulates human conversation through text or voice interactions

Core Components of a Chatbot System

NLU Natural Language Understanding

Parse user input → intent + entities

DM Dialogue Management

Track state, decide next action

NLG Natural Language Generation

Generate human-like responses

KB Knowledge Base

Store facts, FAQs, business logic

Common Use Cases

- Customer Support
- Banking
- E-commerce
- Education
- Healthcare
- Entertainment

Types of Chatbots: Evolution

Gen 1
Rule-Based

1960s-2010s

Pattern matching, decision trees, if-then rules

✓ Pros:

Predictable, easy to debug

✗ Cons:

Limited, brittle, no learning

Examples:

ELIZA, early IVR systems

→

Gen 2
ML-Based (NLU)

2010s-2020

Intent classification + slot filling with ML models

✓ Pros:

Handles variations, learns from data

✗ Cons:

Needs training data, domain-specific

Examples:

Rasa, Dialogflow, Lex

→

Gen 3
LLM-Powered

2020-Present

Large Language Models with prompting/RAG

✓ Pros:

Flexible, general knowledge, natural

✗ Cons:

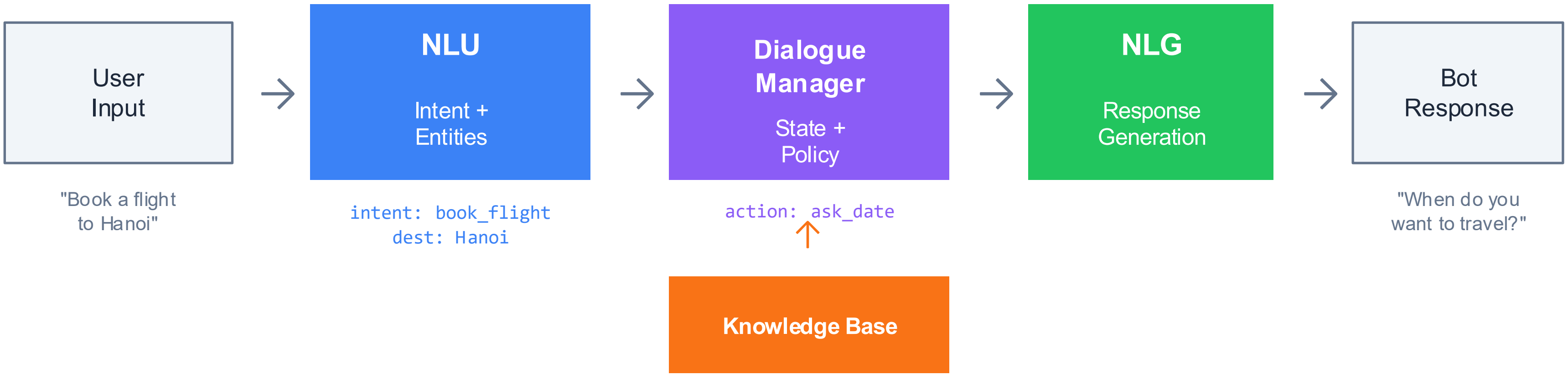
Hallucinations, costly, less control

Examples:

ChatGPT, Claude, Gemini

Traditional Chatbot architecture (pipeline)

Task-Oriented Chatbot Pipeline



User Input → NLU → Dialogue Manager → NLG → Bot Response

"Book a flight to Hanoi"

NLU

Intent + Entities

`intent: book_flight`
`dest: Hanoi`

Dialogue Manager

State + Policy

`action: ask_date`

Knowledge Base

NLG

Response Generation

Bot Response

"When do you want to travel?"

Key Concepts in Pipeline Architecture

Intent: What the user wants (e.g., book_flight, cancel_order)

Entity/Slot: Key information (e.g., destination=Hanoi, date=tomorrow)

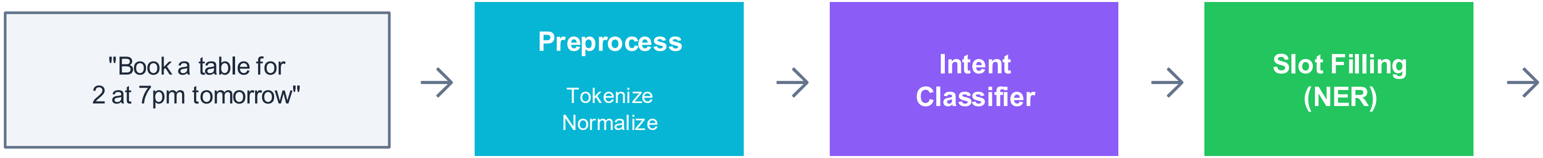
Dialogue State: Current conversation context and filled slots

Policy: Rules/model deciding what action to take next

NLU Pipeline: From text to structured data

NLU transforms unstructured user text into structured representation that machines can process

NLU Processing Pipeline



"Book a table for 2 at 7pm tomorrow" → Preprocess (Tokenize, Normalize) → Intent Classifier → Slot Filling (NER) →

Structured Output:

```
{
  intent: "book_table",
  slots: { num_people: "2", time: "7pm", date: "tomorrow" }
}
```

Intent Classification

Multi-class classification: Map input to one of predefined intents (book_table, cancel, etc.)

Slot Filling (NER)

Sequence labeling: Extract entities and their types (date, time, location, quantity, etc.)

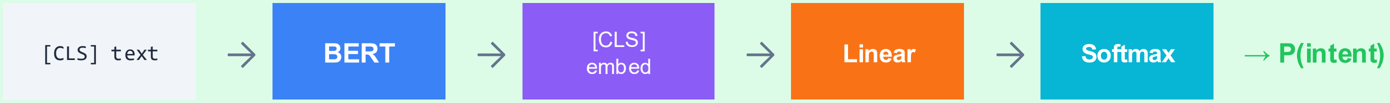
Intent classification: models & methods

Intent Classification = Text Classification → Map user utterance to one of N predefined intent classes

Classification Methods Comparison

Traditional ML	Deep Learning	Transformers
Models: SVM, Naive Bayes, RF	**Models:** CNN, LSTM, BiLSTM	**Models:** BERT, RoBERTa, DistilBERT
Features: TF-IDF, n-grams, BoW	**Features:** Word embeddings	**Features:** Contextual embeddings
✓ Fast, interpretable ✗ Need feature engineering	✓ Learn features automatically ✗ Need more data	✓ SOTA accuracy, transfer learning ✗ Compute intensive

Modern Approach: BERT-based Intent Classification



`[CLS] text` → BERT → [CLS] embed → Linear → Softmax → P(intent)

Fine-tune BERT on your intent dataset — typically 90%+ accuracy with few hundred examples!

Slot Filling: Named Entity Recognition

Slot Filling = Sequence Labeling → Assign a label to each token in the input sequence

BIO Tagging Scheme Example

Tokens:	Book	a	flight	to	Ha	Noi	tomorrow	at	3pm
Labels:	O	O	O	O	B-LOC	I-LOC	B-DATE	O	B-TIME

B = Begin, I = Inside, O = Outside entity

Slot Filling Models

BiLSTM-CRF	Bidirectional LSTM + Conditional Random Field for sequence constraints
BERT + Linear	BERT encoder + linear layer per token, fine-tuned end-to-end
Joint Model	Single model for intent + slots sharing encoder (multi-task learning)

Dialogue State Tracking (DST)

DST maintains a structured representation of the conversation: what slots have been filled, what's still needed

Example Conversation

User: I want to book a restaurant

```
→ intent: book_restaurant
```

Bot: What cuisine do you prefer?

User: Vietnamese food please

```
→ cuisine: vietnamese
```

Bot: For how many people?

User: 4 people, tomorrow night

```
→ num_people: 4, date: tomorrow, time: night
```

Current Dialogue State

```
{
  intent: "book_restaurant",
  slots: {
    cuisine: "vietnamese",
    num_people: 4,
    date: "tomorrow",
    time: "night",
    location: null  // unfilled
  },
  turn_count: 3
}
```

Next Action: location is null → Ask "Where would you like to dine?" OR use default location

LLM-based Chatbots: the new paradigm

Shift: From pipeline components to single powerful model that handles NLU + DM + NLG together!

Traditional Pipeline

```
User Input → NLU → DM → NLG → Response
```

- Separate models for each component
- Need training data for each task
- Errors propagate through pipeline
- Limited to predefined intents/slots

LLM-Based Approach

```
User Input → LLM (+ context) → Response
```

- Single model handles everything
- Few-shot or zero-shot learning
- Flexible, handles new scenarios
- Natural, human-like responses

Key Technologies for LLM Chatbots

RAG
Retrieval-Augmented Generation

Fine-tuning
Adapt LLM to your domain

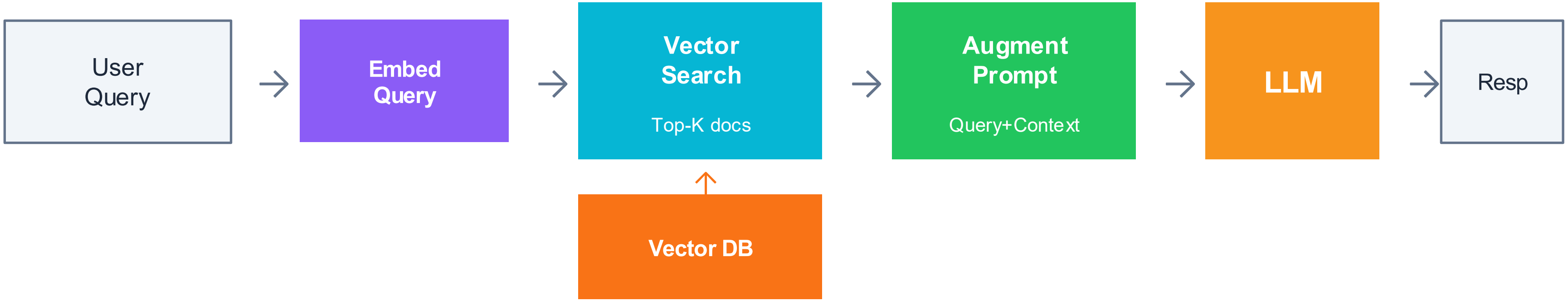
Prompting
In-context learning

Guardrails
Safety & constraints

RAG: Retrieval-Augmented Generation

RAG = Retrieve relevant documents + Augment prompt with context + Generate response with LLM

RAG Pipeline Architecture



RAG Benefits

- Up-to-date knowledge
- Reduced hallucinations
- Domain-specific answers

Popular Tools

- Vector DBs: Pinecone, Chroma, Weaviate
- Frameworks: LangChain, LlamaIndex

Fine-tuning vs Prompting: When to Use What?

Fine-tuning

What:
Train model weights on your data

When to use:
- Specific domain/style needed
- Have labeled training data (1k+)
- Need consistent behavior

Pros:
Better performance on specific tasks

Cons:
Costly, needs data, may overfit

Prompting (In-Context Learning)

What:
Guide model via instructions/examples

When to use:
- Quick prototyping
- Limited/no training data
- Need flexibility

Pros:
Fast iteration, no training needed

Cons:
Limited by context window, variable

Decision Guide

Start with prompting → If not enough, try RAG → If still not enough, fine-tune → For best results, combine all!

Prompt engineering for Chatbots

System Prompt Structure

Role:

You are a helpful customer support agent for TechCorp...

Instructions:

Answer questions about products. Be polite. Ask clarifying questions if needed...

Constraints:

Do not discuss competitors. Do not make promises about refunds...

Format:

Keep responses under 100 words. Use bullet points for lists...

Examples:

User: How do I reset? Assistant: To reset your device, go to Settings > Reset...

Best Practices

- Be specific and clear
- Use delimiters (###, """)
- Give examples (few-shot)
- Specify output format
- Test edge cases
- Iterate and refine

Advanced Techniques

Chain-of-Thought:	Step-by-step reasoning
Self-Consistency:	Multiple paths, vote
ReAct:	Reason + Act iteratively
Reflection:	Self-critique & improve

Chatbot evaluation metrics

Task-Specific Metrics

Intent Classification

Accuracy, F1-score, Precision, Recall

Slot Filling (NER)

Entity-level F1, Span-level accuracy

Dialogue Success

Task completion rate, # turns to complete

Response Quality Metrics

BLEU:	N-gram overlap with reference (translation)
ROUGE:	Recall-oriented for summarization
BERTScore:	Semantic similarity using BERT embeddings
Perplexity:	How well model predicts responses

Human Evaluation (Most Important!)

Fluency	Is the response grammatically correct and natural?	[1-5]	**Relevance**	Does it address the user's question/request?	[1-5]
Helpfulness	Did it actually help the user achieve their goal?	[1-5]	**Safety**	Is the response appropriate and harmless?	[Yes/No]

Common pitfalls & solutions

❌ Hallucinations

LLM generates plausible but false information

✓ Use RAG, add fact-checking, constrain outputs

❌ Context Loss

Bot forgets earlier conversation turns

✓ Include conversation history in prompt, use memory

❌ Off-topic Responses

Bot responds to irrelevant or harmful requests

✓ Add guardrails, intent filtering, safety checks

❌ Inconsistent Persona

Bot's personality/knowledge varies across turns

✓ Strong system prompt, fine-tuning, testing

❌ Latency Issues

Slow response times frustrate users

✓ Use smaller models, caching, streaming responses

❌ Handling Errors

Bot crashes or gives unhelpful error messages

✓ Graceful fallbacks, human handoff, logging

Summary: Chatbot Development

1 **Chatbot Types**

Rule-based → ML/NLU → LLM-powered evolution

2 **NLU Pipeline**

Intent classification + Slot filling + Dialogue state

3 **LLM Chatbots**

Single model for NLU+DM+NLG, more flexible

4 **RAG**

Retrieve context + Augment prompt + Generate

5 **Prompt Engineering**

Role, instructions, constraints, format, examples

Tools & Frameworks

Models:

`BERT, GPT, LLaMA, Mistral`

Libraries:

`HuggingFace, LangChain`

Platforms:

`Rasa, Dialogflow, Amazon Lex`

Vector DBs:

`Pinecone, Chroma, Weaviate`

Evaluation

Measuring Success

- Automatic Evaluation Metrics (BLEU, ROUGE, BERTScore)
- Human Evaluation Frameworks
- A/B Testing & Online Evaluation
- Standard Benchmarks & Datasets
- Hands-on Implementation Lab

Why evaluation matters

"You can't improve what you don't measure" — Evaluation guides development, deployment, and iteration

Automatic Metrics

What:
Computed automatically from outputs

Examples:
BLEU, ROUGE, F1, BERTScore

Pros:
Fast, scalable, reproducible

Human Evaluation

What:
Human judges rate quality

Examples:
Likert scales, A/B preference

Pros:
Captures nuance, real quality

Online Evaluation

What:
Real user behavior metrics

Examples:
Task success, engagement, CTR

Pros:
Real-world performance

Evaluation Pipeline: Offline (Auto + Human) → Online A/B Test → Production Monitoring

Best practice: Use automatic metrics for fast iteration, human eval for quality gates, online metrics for deployment decisions

Automatic Evaluation Metrics

BLEU (Bilingual Evaluation Understudy)

$$\mathrm{BLEU} = \mathrm{BP} \times \exp(\Sigma\, w_n \log p_n)$$

p_n = n-gram precision
BP = brevity penalty (penalizes short outputs)
Typically BLEU-4 uses n = 1,2,3,4

ROUGE (Recall-Oriented Understudy)

$$\text{ROUGE-N} = |\text{matched n-grams}| \;/\; |\text{ref n-grams}|$$

ROUGE-1: unigram recall
ROUGE-2: bigram recall
ROUGE-L: longest common subsequence

F1 Score (Intent & Slot Evaluation)

$$F1 = 2 \times (P \times R) \;/\; (P + R)$$

P = Precision = TP / (TP + FP)
R = Recall = TP / (TP + FN)
Use macro/micro F1 for multi-class

BERTScore (Semantic Similarity)

$$\text{BERTScore} = \text{cosine\_sim}(\text{BERT}(\text{cand}), \text{BERT}(\text{ref}))$$

Uses BERT embeddings for semantic matching
Captures meaning even with different wording
More robust than n-gram metrics

Use: BLEU for translation | ROUGE for summarization | F1 for classification/NER | BERTScore for open-ended generation

Human Evaluation Framework

Human evaluation is the gold standard — automatic metrics don't capture all aspects of quality

Common Evaluation Dimensions

Fluency	Is the response grammatically correct and natural?	[1-5]
Relevance	Does it address the user's actual question?	[1-5]
Correctness	Is the information factually accurate?	[1-5]
Helpfulness	Did it help achieve the user's goal?	[1-5]
Safety	Is it appropriate and harmless?	[Yes/No]

Evaluation Methods

Likert Scale Rating
Rate each response 1-5

Pairwise Comparison
"Which response is better?"

Best-Worst Scaling
Pick best & worst from N options

Side-by-Side (SxS)
Compare A vs B directly

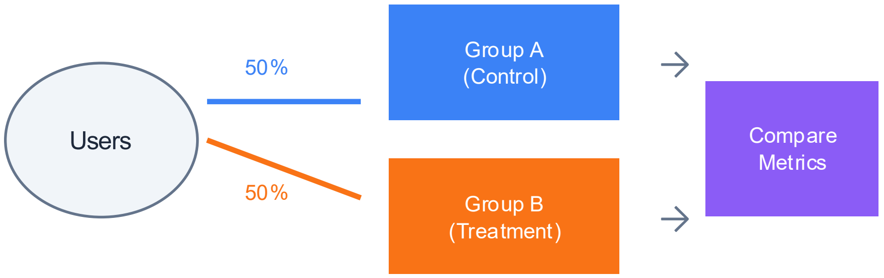
Best Practices

• Use 3+ annotators per item • Calculate inter-annotator agreement (Kappa ≥ 0.6) • Randomize order • Blind to system identity • Clear guidelines + examples

A/B Testing & Online Evaluation

A/B Testing: Compare two versions with real users to make data-driven decisions

A/B Test Setup



Key Metrics to Track:

- Task completion rate
- User satisfaction (thumbs up/down)
- Session length, # of turns
- Fallback/error rate

Statistical Considerations

Sample Size
Need enough users for significance
(typically 1000+ per variant)

Statistical Significance
p-value < 0.05 (95% confidence)

Effect Size
How big is the improvement?
Cohen's d, relative lift %

Decision Framework

B significantly better → Deploy B | No significant difference → Consider cost/complexity | B worse → Keep A, investigate why

Standard Benchmarks & Datasets

NLU Benchmarks

GLUE/SuperGLUE	General language understanding
SNIPS	Intent classification + slot filling
ATIS	Airline travel domain NLU
MultiWOZ	Multi-domain dialogue state
CoNLL-2003	Named Entity Recognition

Generation Benchmarks

PersonaChat	Persona-based conversation
DailyDialog	Daily life conversations
Wizard of Wikipedia	Knowledge-grounded dialogue
ConvAI2	Open-domain chatbot challenge
MT-Bench	Multi-turn LLM evaluation

LLM Leaderboards & Evaluation Suites

LMSYS Chatbot Arena	Human preference voting, ELO ranking	Open LLM Leaderboard	HuggingFace benchmark suite
HELM	Stanford holistic LLM evaluation	AlpacaEval	GPT-4 based automatic evaluation

Summary: Evaluation & Implementation

1 **Automatic Metrics**
BLEU, ROUGE, F1, BERTScore — fast & scalable

2 **Human Evaluation**
Fluency, relevance, helpfulness — gold standard

3 **A/B Testing**
Real users, statistical significance, data-driven decisions

4 **Benchmarks**
SNIPS, MultiWOZ, MT-Bench — standardized comparison

Key Formulas

BLEU:
$BP \times \exp(\Sigma\ w_n \log p_n)$

F1 Score:
$2\times(P\times R)/(P+R)$

BERTScore:
$\text{cosine}(\text{BERT}(c), \text{BERT}(r))$

Kappa (Agreement):
$(p_o - p_e)/(1 - p_e)$

Assignment & Summary

Build Your Own Transformer-based Chatbot

Assignment: Build a transformer chatbot

Final Project: End-to-End Chatbot Development

Apply Attention, Transformers, and NLU concepts to build a working chatbot system

Learning Objectives

- Understand Transformer architecture in practice
- Implement intent classification with BERT
- Build slot filling / NER system
- Design dialogue management logic
- Evaluate using appropriate metrics
- Document and present your work

Team Formation

- Individual or teams of 2-3 students
- Larger teams require proportionally larger scope
- Clear contribution statement required

Session summary

Part 1

Introduction & Attention

Seq2Seq bottleneck, Attention mechanism basics

Part 2

Self-Attention & Multi-Head

Q,K,V framework, Scaled dot-product, MHA

Part 3

Transformer Architecture

Positional encoding, Encoder, Decoder, BERT/GPT/T5

Part 4

Chatbot Development

NLU pipeline, Intent+Slot, RAG, Prompt engineering

Part 5

Evaluation & Implementation

BLEU, ROUGE, F1, Human eval, A/B testing, Lab

Part 6

Assignment & Summary

Project requirements

Key formulas review

Attention Mechanisms

Scaled Dot-Product:

$\mathrm{Attn}(Q,K,V) = \mathrm{softmax}(QK^T/\sqrt{d_k})\cdot V$

Multi-Head Attention:

$\mathrm{MHA} = \mathrm{Concat}(\mathrm{head}_1,\ldots,\mathrm{head}_h)\cdot W^O$

Linear Projections:

$Q=XW^{\psi},\ K=XW^K,\ V=XW^V$

Positional Encoding:

$PE(pos,2i) = \sin(pos/10000^{\wedge}(2i/d))$

Transformer Components

Feed-Forward Network:

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

Layer Normalization:

$\mathrm{LN}(x) = \gamma\cdot(x-\mu)/\sigma + \beta$

Residual Connection:

$\mathrm{output} = \mathrm{LayerNorm}(x + \mathrm{Sublayer}(x))$

Masked Attention:

$M[i,j] = 0 \text{ if } j\leq i, \text{ else } -\infty$

Evaluation Metrics

BLEU:	$BP \times \exp(\Sigma\ w_n \log p_n)$	**ROUGE-N:**	$\lvert\text{matched n-grams}\rvert / \lvert\text{ref n-grams}\rvert$
F1 Score:	$2\times(P\times R)/(P+R)$	**BERTScore:**	$\mathrm{cosine}(\mathrm{BERT}(cand), \mathrm{BERT}(ref))$