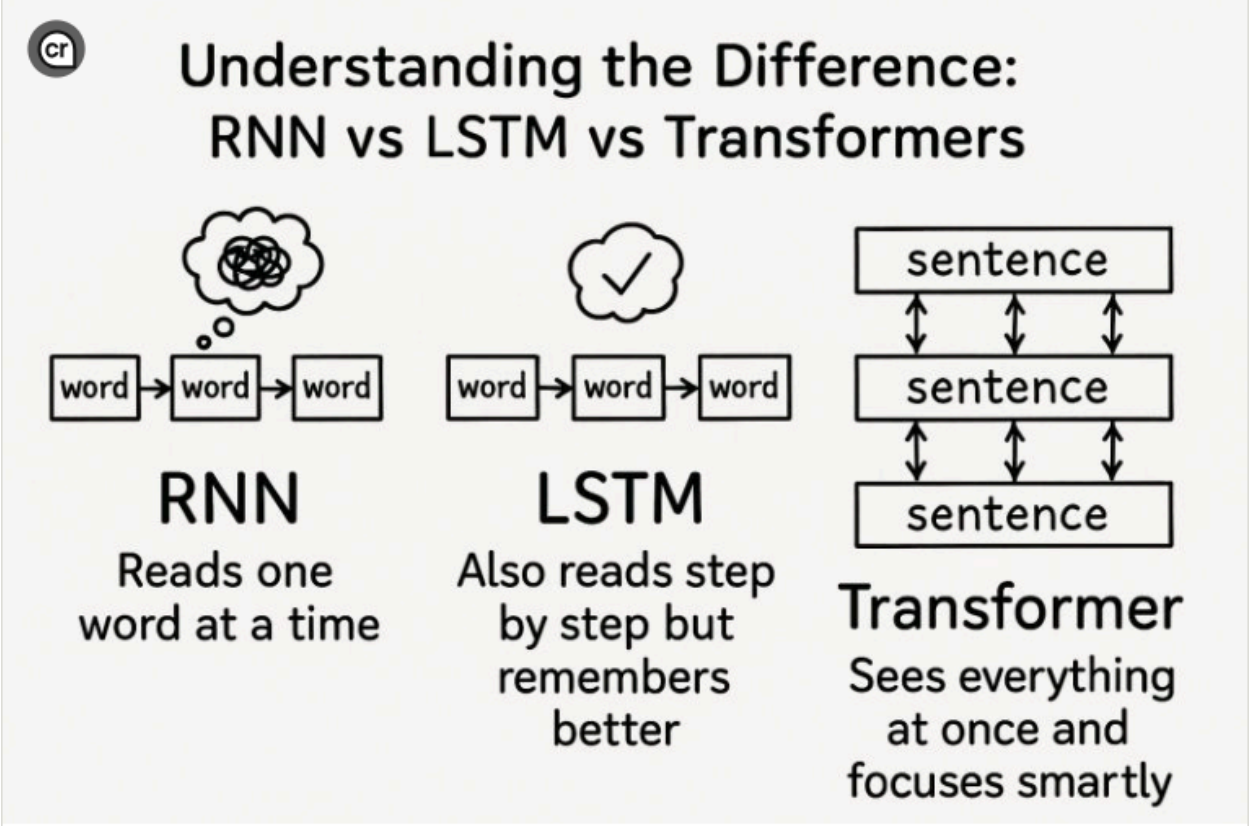
Transformer Architecture: LLM-এর ভিত্তি

১) ট্রান্সফর্মার কেন এত গুরুত্বপূর্ণ?

ট্রান্সফর্মার হলো এমন একটি নিউরাল নেটওয়ার্ক আর্কিটেকচার যা **sequence (**টেক্সট/টোকেন**)** প্রসেস করতে পারে এবং একই সাথে দূরের শব্দের সম্পর্ক (long-range dependency) ধরতে পারে—এবং এটি RNN/LSTM-এর মতো একে একে শব্দ প্রসেস না করে প্যারালালভাবে কাজ করতে পারে। এই আর্কিটেকচার প্রথম জনপ্রিয় হয় ২০১৭ সালের "Attention Is All You Need" গবেষণাপত্রের মাধ্যমে। (arXiv)

কেন বিপ্লব**?**

- Self-attention ব্যবহার করে দূরের টোকেনের সম্পর্ক বোঝে
- Training দ্রুত (parallelizable)
- স্কেল করলে ক্ষমতা নাটকীয়ভাবে বাড়ে (আজকের LLM যুগের ভিত্তি)



২) ট্রান্সফর্মারের জন্ম ও মূল উদ্দেশ্য

ট্রান্সফর্মার মূলত **sequence-to-sequence** কাজের জন্য এসেছে—বিশেষ করে **machine translation** (যেমন English → German/French)। “Attention Is All You Need” পেপারেই এই লক্ষ্য ও ফলাফল দেখানো হয়েছে। (arXiv)

এখান থেকে পরে দুইটা বড় পরিবার বের হয়:

1. **Encoder-based** (যেমন BERT)
2. **Decoder-only** (যেমন GPT পরিবার—next token prediction)

৩) Transformer কীভাবে কাজ করে?

Transformer বোঝার সবচেয়ে সহজ ফ্রেম:
Text → Token → Vector → Attention → New Vector → Output

Step 1: Input text

তুমি একটি বাক্য দিলে:
“আমি আজ স্কুলে যাব”

Step 2: Tokenization

বাক্যটাকে ভাঙা হয় টোকেনে (শব্দ/শব্দাংশ)। প্রতিটা টোকেনের একটা ID থাকে।

```
Text → [Token1, Token2, …] → [ID1, ID2, …]
```

Step 3: Token Embedding

প্রতিটা token ID কে একটি ঘন ভেক্টরে (dense vector) রূপান্তর করা হয়।
এটা এমন এক space যেখানে অর্থগতভাবে কাছাকাছি শব্দ কাছাকাছি অবস্থান করে—এটাই semantic representation.

```
Token IDs → Embedding lookup → Vectors (d_model)
```

Step 4: Positional Encoding

Transformer নিজে থেকে “ক্রম/পজিশন” জানে না (কারণ এটি parallel)। তাই প্রতিটা token vector-এর সাথে **position information** যোগ করা হয়—যাতে “প্রথম শব্দ”, “দ্বিতীয় শব্দ” ইত্যাদি বোঝা যায়।

```
Embedding + Positional Encoding → Position-aware vectors
```

৪) Self-Attention: Transformer-এর “প্রাণ”

Self-Attention কী কাজ করে?

Self-attention প্রতিটা token-কে শেখায়:

> “এই বাক্যে আমার জন্য কোন কোন শব্দ সবচেয়ে বেশি গুরুত্বপূর্ণ?”

উদাহরণ:
“I saw the man with the telescope.”
এখানে “with the telescope” কার সাথে যুক্ত—“I saw” নাকি “the man”?
Self-attention এই ধরনের সম্পর্ক ধরতে সাহায্য করে।

Long-range dependency

Self-attention দূরের টোকেনকেও গুরুত্ব দিতে পারে। পেপারে দেখানো হয়েছে transformer recurrence ছাড়াই attention দিয়ে কাজ করে। (arXiv)

Key, Query, Value (K, Q, V) — সহজ ব্যাখ্যা

Transformer প্রতিটি token vector থেকে তিনটি জিনিস বানায়:

- **Query (Q):** “আমি কী খুঁজছি?”
- **Key (K):** “আমার মধ্যে কী তথ্য আছে?”
- **Value (V):** “আমার আসল তথ্য (যা অন্যরা নেবে)”

তারপর:

- Query একটি token-এর
- Key অন্য token গুলোর
  মিলে একটা similarity score বের হয় → এই স্কোর অনুযায়ী Value গুলো weighted sum হয়ে নতুন representation তৈরি করে।

```
Q × Kᵀ → attention weights → weights × V → new token vector
```

Scaled Dot-Product Attention (কেন “scaled”?)

Dot-product বড় হলে softmax saturate হতে পারে, তাই স্কেল করা হয়—এটা “Attention Is All You Need” পেপারের মূল অংশ। (NeurIPS Proceedings)

Multi-Head Attention

একটা attention head মানে “একটা দৃষ্টিকোণ”। Multi-head মানে একই বাক্যকে একাধিক দৃষ্টিকোণ থেকে দেখা:

- এক head grammar সম্পর্ক ধরতে পারে
- আরেক head subject–verb agreement
- আরেক head coreference (he → who?)

শেষে সব head একসাথে combine হয়।

```
Multi-head attention: Head1, Head2, … → concat → linear
```

৫) Transformer Block-এর ভেতরের সম্পূর্ণ কাঠামো

একটি standard Transformer layer/block সাধারণত:

1. Multi-Head Self-Attention
2. Add & LayerNorm (Residual connection)
3. Feed Forward Network (FFN)
4. Add & LayerNorm

Residual connection কেন?
Training stable রাখতে এবং gradient flow ভালো করতে।

```
MHA → Add&Norm → FFN → Add&Norm
```

৬) Encoder–Decoder বনাম Decoder-only (LLM-এর সাথে সংযোগ)

(A) Encoder–Decoder Transformer (Translation style)

- Encoder ইনপুট বাক্যকে representation করে
- Decoder একে একে আউটপুট তৈরি করে
  এটাই মূল ট্রান্সফর্মার (original paper) কাঠামো। (arXiv)

```
Input → Encoder → Context → Decoder → Output
```

(B) Decoder-only Transformer (GPT style)

LLM-এর বড় অংশ (ChatGPT টাইপ) হলো **decoder-only**:

- শুধু আগের টোকেন দেখে পরের টোকেন predict করে (causal / left-to-right)
- attention mask থাকে, যাতে future token দেখা না যায়

```
Tokens (left context only) → masked self-attention → next token prediction
```

৭) BERT বনাম GPT — transformer ভিত্তি একই, উদ্দেশ্য আলাদা

BERT (Encoder-only, bidirectional)

BERT হলো **bidirectional**: একসাথে left এবং right context দেখে representation শেখে। সাধারণ pretraining objective: **Masked Language Modeling** (মাঝখানের mask করা টোকেন অনুমান)। (arXiv)

কাজে ভালো:

- classification
- sentiment analysis
- extractive QA (প্রসঙ্গ থেকে উত্তর বের করা)

GPT (Decoder-only, causal)

GPT পরিবার সাধারণত **next token prediction** করে, left-to-right। এটাই text generation-এর ভিত্তি।

৮) “Transformer” বনাম “LLM” — ভুল ধারণা পরিষ্কার

এইটা খুব গুরুত্বপূর্ণ:

- সব **Transformer LLM** নয়
  কারণ Transformer Vision/Audio/Multimodal—সব জায়গায় ব্যবহার হয়। উদাহরণ: **Vision Transformer (ViT)** ইমেজকে patch হিসেবে token বানিয়ে transformer দিয়ে classify করে। (arXiv)
- সব **LLM-ই Transformer** নয় **(**ইতিহাসগতভাবে**)**
  Transformer আসার আগে বড় ভাষা মডেলগুলো RNN/LSTM ভিত্তিক ছিল। (আজকের আধুনিক LLM-এ transformer ডমিন্যান্ট, কিন্তু ধারণাগতভাবে LLM = “বড় ভাষা মডেল”, আর্কিটেকচার ভিন্ন হতে পারে।)

“Transformer applications: NLP + Vision (ViT) + others”

৯) ‘Key Points’

1. Transformer ২০১৭ সালের “Attention Is All You Need” থেকে জনপ্রিয় হয় এবং self-attention ব্যবহার করে recurrence ছাড়াই sequence modeling করে। (arXiv)
2. Self-attention = প্রতিটি token বাক্যের অন্য token গুলোর সাথে সম্পর্ক শিখে।
3. Q-K-V + scaled dot-product + multi-head = attention-এর core। (NeurIPS Proceedings)
4. BERT = encoder-only, bidirectional masked LM। (arXiv)
5. GPT-style = decoder-only, causal next-token generation।
6. Transformer শুধু টেক্সট নয়—ViT দেখায় ইমেজেও transformer কাজ করে। (arXiv)

A) Transformer Cheat-Sheet (১ পেজ নোট)

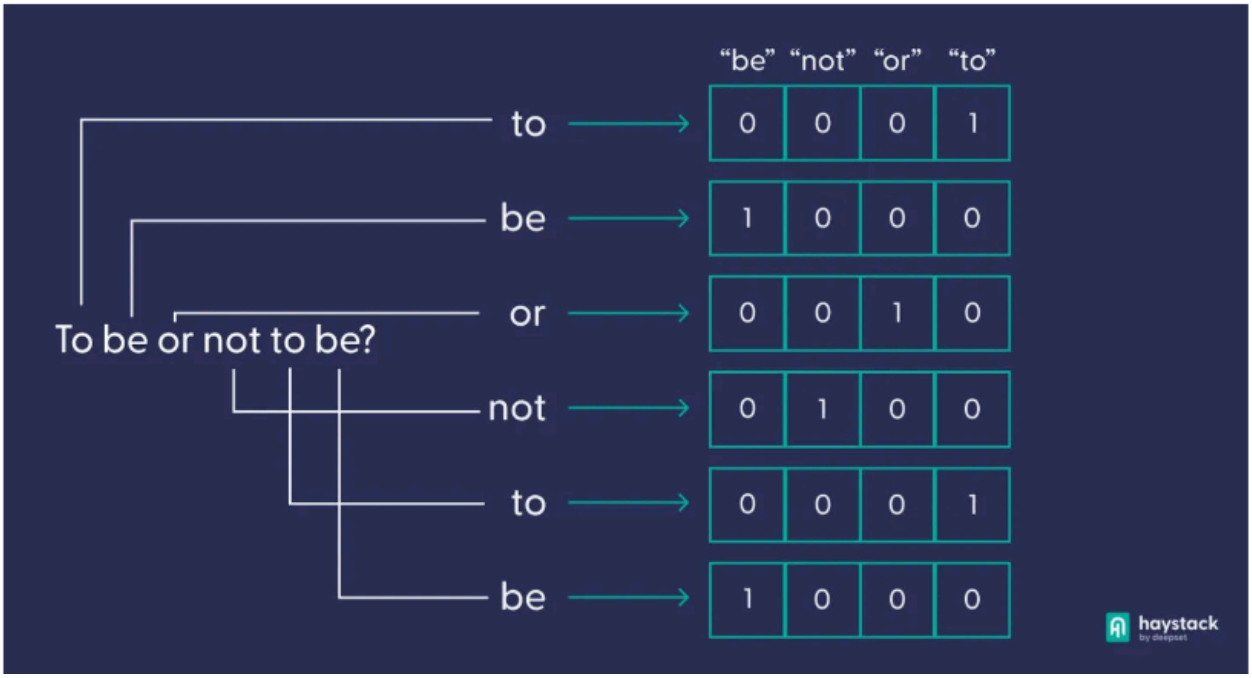
0) Big Picture

Transformer = Attention-based sequence model

- RNN/LSTM-এর মতো sequential না; **parallel** প্রসেস করতে পারে
- Core ধারণা: **Self-Attention** (token ↔ token সম্পর্ক)

1) Input Pipeline (Text → Vectors)

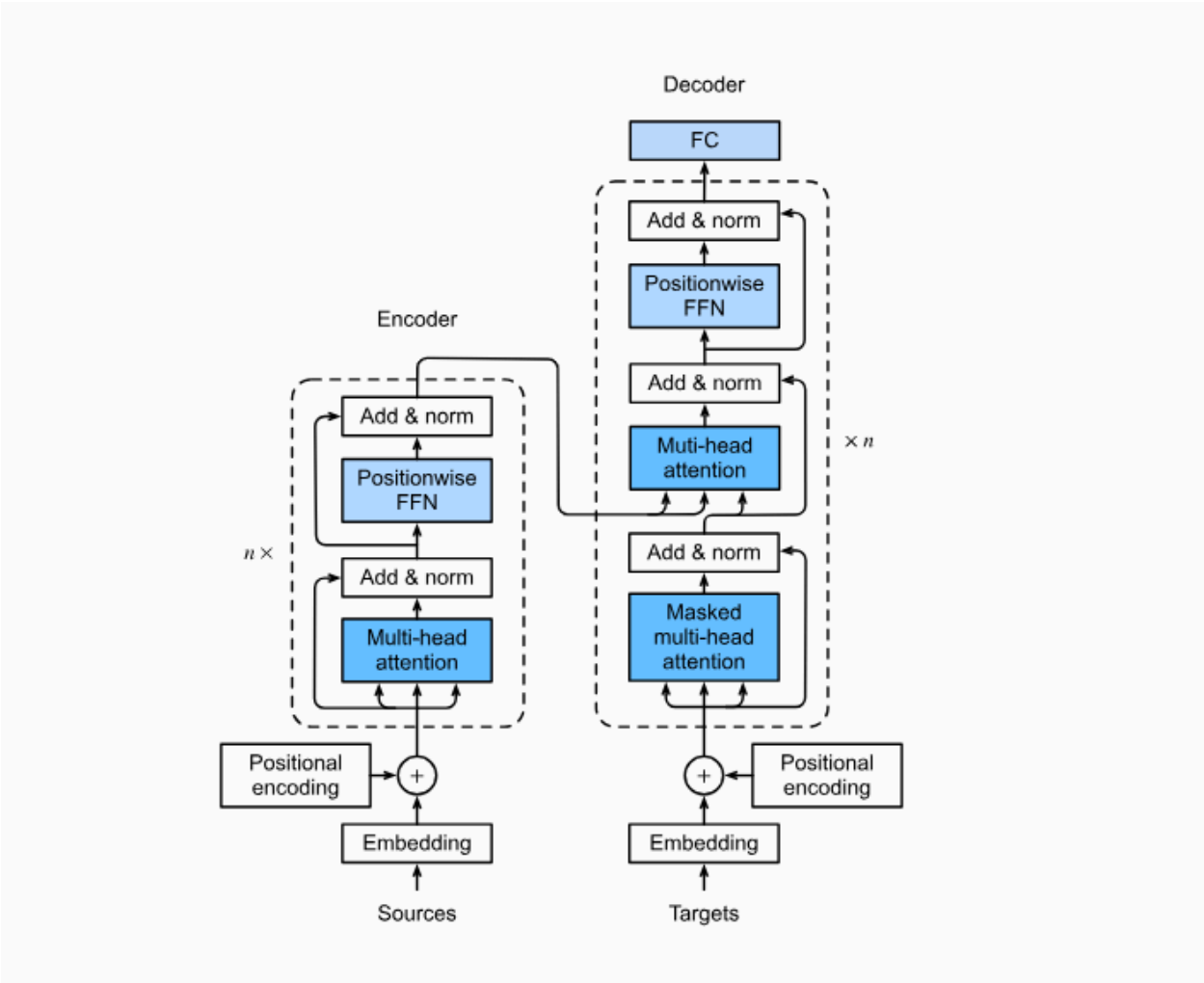
1. **Tokenization**: text → token IDs
2. **Embedding**: token IDs → dense vectors (d_model)
3. **Positional Encoding**: order বোঝাতে embedding-এ position info যোগ



2) Transformer Block (একটি লেয়ার)

একটি standard block সাধারণত:

1. **Multi-Head Self-Attention (MHA)**
2. **Add & LayerNorm** (Residual connection)
3. **Feed-Forward Network (FFN)**
4. **Add & LayerNorm**



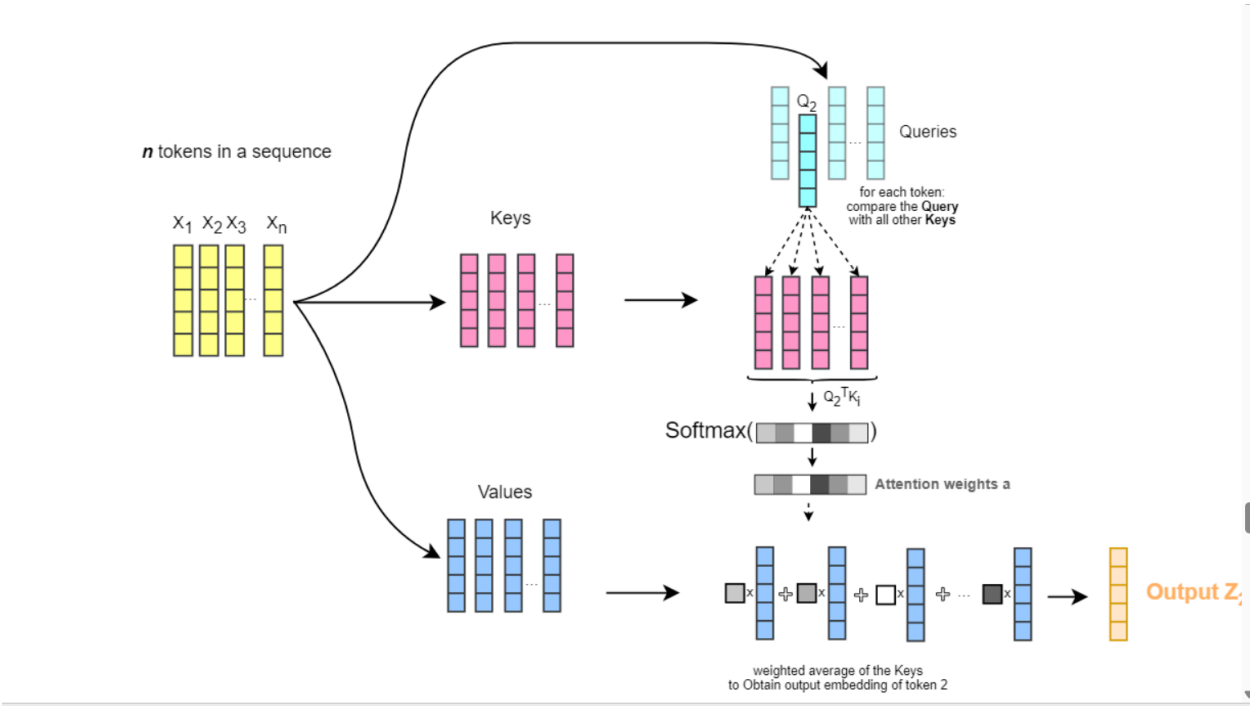
3) Self-Attention Core (Q, K, V)

প্রতি token vector থেকে তৈরি হয়:

- **Q (Query)**: “আমি কী খুঁজছি?”
- **K (Key)**: “আমার ভেতরে কী সংকেত আছে?”
- **V (Value)**: “আসল তথ্য যা অন্যরা নেবে”

Attention scores:

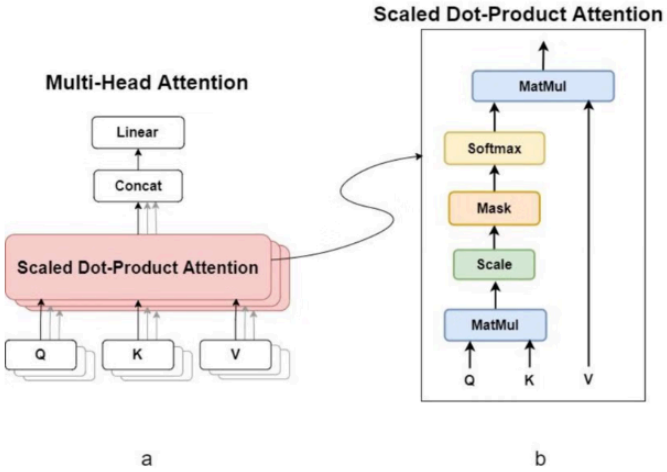
- Score = $Q \cdot K^T$ (scaled) → softmax → weights
- Output = weights × V (weighted sum)



4) Multi-Head Attention (MHA)

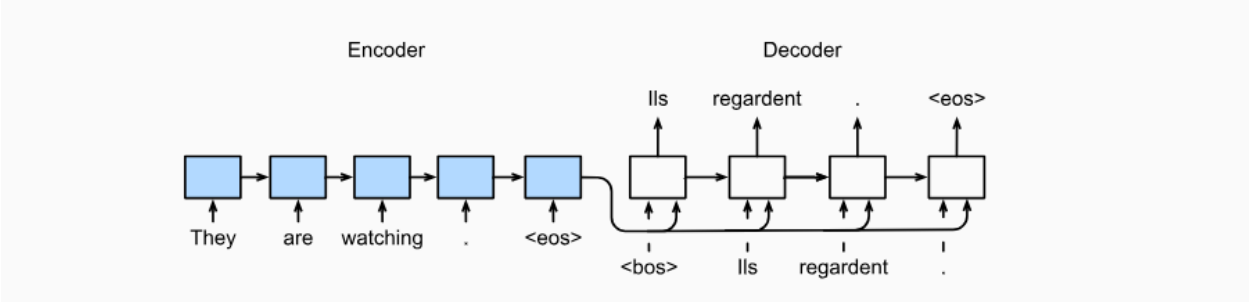
একাধিক head = একাধিক “view”

- Head1 grammatical relation ধরতে পারে
- Head2 coreference (he → who?) ধরতে পারে
- Head3 long dependency ধরতে পারে
  সব head concat হয়ে final projection



5) Encoder vs Decoder vs Decoder-Only

- **Encoder-Decoder**: translation style (original Transformer)
- **Encoder-only**: BERT-style (representation learning; MLM)
- **Decoder-only**: GPT-style (generation; causal LM)



---

6) BERT vs GPT (Quick contrast)

- **BERT**: bidirectional context, masked tokens predict → classification/understanding
- **GPT**: left-to-right next token predict → generation/chat

Figure A7 (Placeholder): "BERT vs GPT"
Caption: *BERT uses bidirectional encoding; GPT uses causal decoding for next-token generation.*

---

7) Transformer vs LLM (Common confusion)

- সব Transformer LLM নয় (Vision Transformer ইমেজে কাজ করে)
- সব LLM ঐতিহাসিকভাবে Transformer ছিল না (আগে RNN/LSTM ছিল)

B) Self-Attention Deep Diagram (Hand-draw guide)

এটা তোমার নোটে সবচেয়ে "must-have" অংশ। আমি একদম হাতে আঁকার মতো করে দিলাম—তুমি শুধু box/arrow কপি করলেই হবে।

B1) Self-Attention: What it computes

ধরা যাক তোমার কাছে 3 token আছে:
`t1, t2, t3`
এদের embedding (positional যোগ করা) হলো:
`X = [x1, x2, x3]` যেখানে প্রতিটা xi হলো d_model ডাইমেনশনের ভেক্টর

এখন তিনটা projection:

- **Q = X Wq**
- **K = X Wk**
- **V = X Wv**

Attention formula (single head)

1. Scores: $S = (Q K^T) / \sqrt{d\_k}$
2. Weights: $A = softmax(S)$ (row-wise)
3. Output: $O = A V$

B2) Hand-draw Diagram (Step-by-step)

Step 1: Input vectors

একটা বড় box আঁকো: **X (n×d_model)**
ভেতরে 3টা row দেখাও: x1, x2, x3

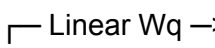
Figure B1 (draw):
```
[x1]
[x2]
[x3]
```

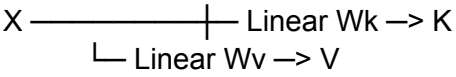
Step 2: Three linear layers to get Q, K, V

X থেকে তিনটা শাখা:

- X → Linear(Wq) → Q
- X → Linear(Wk) → K
- X → Linear(Wv) → V

Figure B2 (draw):





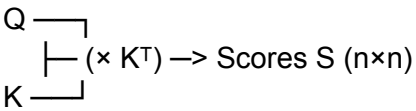
Caption: *Same input X is projected into queries, keys, and values.*

Step 3: Score calculation QK

Q কে K^T এর সাথে multiply:
`Q (n×d_k)` × `K^T (d_k×n)` = `S (n×n)`
এখানে S হলো “token-to-token relevance matrix”

- S[i,j] মানে: token i কতটা token j-কে attend করবে

Figure B3 (draw):



Extra note (লিখে দেবে**):** `/ √d_k` scaling

Step 4: softmax → attention weights

Scores S এর প্রতিটা row-তে softmax দিলে weights A পাওয়া যায়।
Row-wise softmax মানে:

- প্রতিটা token i অন্য token গুলোর ওপর probability distribution বানায়
- প্রতিটা row এর sum = 1

```
Scores S → softmax → A (attention weights)
```

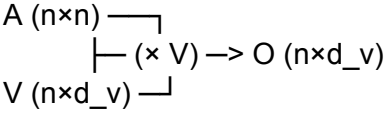
Step 5: Weighted sum of V

এখন A কে V দিয়ে multiply করলে output O পাওয়া যায়:

```
O = A V
```

O-র প্রতিটা row হলো token i-এর “new representation” (context-aware).

Figure B5 (draw):



Caption: *Each output vector is a weighted mixture of value vectors.*

B3) Tiny numeric intuition (নোটে এক লাইনে)

- যদি token1, token3-এর সাথে বেশি সম্পর্কিত হয় → A[1,3] বড় হবে
- তাহলে output o1 তে v3-এর contribution বেশি হবে

B4) Causal Mask (GPT-style) — কেন লাগে

Decoder-only GPT তে future token দেখা নিষিদ্ধ। তাই attention score matrix S-এ mask বসে:

- token i শুধু ≤ i পর্যন্ত attend করতে পারবে
- upper triangle অংশ -∞ করে softmax দিলে শূন্য হয়ে যায়

Figure B6 (draw):
একটা 3×3 matrix এ upper triangle cross করে দাও:

[✓ x x]
[✓ ✓ x]
[✓ ✓ ✓]

Caption: *Causal mask prevents attending to future tokens.*

B5) Multi-Head Attention-এ Self-Attention কীভাবে বসে

Self-attention (উপরে যা করলে) সেটা একাধিক **head** এ parallel হয়:

Head1: O1, Head2: O2, … → concat → Linear → final

SelfAttn_head1 + SelfAttn_head2 + ... → Concat → Linear → MHA output