Assignment 6 — Solutions

Modern Transformer Variants & BERT

Part B — Transformers & LLMs · 18 questions with answers

Q1. Give examples of encoder-only, decoder-only and encoder–decoder models. Which tasks suit each? ★ · Asked: TCS

Answer

- **Encoder-only:** BERT, RoBERTa, DistilBERT, ALBERT, DeBERTa, ELECTRA; embedding models (BGE, E5). Best for **understanding** tasks – classification, NER, extractive QA, embeddings/semantic search.
- **Decoder-only:** GPT-2/3/4, Llama, Mistral, Gemma, Qwen, Falcon, Claude. Best for **generation** – chat, completion, code, reasoning; also any task framed as text generation.
- **Encoder–decoder:** T5, Flan-T5, BART, mBART, Pegasus, Whisper. Best for **input → output transformation** – translation, summarization, speech-to-text.

Q2. Why are most modern LLMs decoder-only?

Answer

- **Simple, unified objective:** next-token prediction works on any raw text – no labels or special masking needed; every token gives a training signal.
- **Scales extremely well** – the same architecture keeps improving with more data and parameters.
- **Generation is natural** – one stack does both "understanding" (the prompt) and producing output; any task can be framed as text-in/text-out (in-context learning, instruction following).
- **Efficient inference** with KV caching.
- Simpler engineering than maintaining separate encoder and decoder stacks.

Q3. Explain Multi-Head Attention (MHA), Multi-Query Attention (MQA) and Grouped-Query Attention (GQA). ★

Answer

- **MHA (Multi-Head Attention):** every head has its **own Q, K and V** projections. Highest quality, but at inference the **KV cache** stores K and V for every head → large memory.
- **MQA (Multi-Query Attention):** all heads have their own Q, but **share a single K and a single V head**. KV cache shrinks by the number of heads (e.g. 32×) → much faster decoding, but some quality loss and training instability. (PaLM, Falcon.)
- **GQA (Grouped-Query Attention):** a middle ground – query heads are split into **G groups; each group shares one K/V head**. E.g. 32 query heads, 8 KV heads → 4× smaller KV cache with quality close to MHA. Used in **Llama 2-70B, Llama 3, Mistral**. GQA with G = 1 is MQA; G = number of heads is MHA.

Q4. Which of MHA, MQA and GQA uses the least memory, and which gives the best quality? What is the trade-off?

Answer

- **Memory (KV cache), least to most: MQA < GQA < MHA**.
- **Quality, best to worst: MHA ≥ GQA > MQA** (GQA is very close to MHA).

Trade-off: fewer K/V heads → smaller KV cache, higher throughput, longer context and bigger batches on the same GPU, but less representational capacity. GQA is the popular sweet spot.

Q5. What is the KV cache and why is it needed? How does it affect memory at long context lengths? ★

Answer

KV cache: during autoregressive generation, each new token must attend to all previous tokens. Without caching, the model would **recompute keys and values for the entire prefix at every step** (quadratic waste). The KV cache **stores the K and V tensors of all previous tokens for every layer**, so at each step only the new token's Q, K, V are computed and the new K, V are appended.

Benefit: generation becomes much faster (linear work per new token instead of reprocessing everything).

Memory cost: KV cache size = 2 (K and V) × layers × KV heads × head_dim × sequence length × batch size × bytes per value. Example: Llama-2-7B (32 layers, 32 heads × 128 dim, FP16) ≈ **0.5 MB per token** → a 4,096-token context needs ~2 GB per sequence. At long contexts the KV cache can be larger than the model weights; solutions are GQA/MQA, KV-cache quantization, PagedAttention (vLLM), sliding windows.

Q6. What is Flash Attention and why is it faster without changing the result?

Answer

Flash Attention is an **IO-aware, exact** attention algorithm. Standard attention writes the full n × n score matrix to slow GPU memory (HBM) and reads it back for softmax and for multiplying with V – memory traffic, not maths, is the bottleneck.

Flash Attention:

- **Tiles** Q, K, V into blocks that fit in fast on-chip **SRAM**.
- Computes softmax **incrementally (online softmax)** block by block, and **fuses** all steps into one GPU kernel, so the n × n matrix is never stored in HBM.
- Recomputes attention in the backward pass instead of storing it.

Result: 2–4× faster, memory goes from $O(n^2)$ to **O(n)**, enabling much longer contexts. The output is **mathematically identical** to standard attention – it is not an approximation; only the order of computation changes.

Q7. What is sliding-window attention? How does it still capture global context?

Answer

Sliding-window attention (SWA): each token attends only to the **previous W tokens** (e.g. W = 4,096 in Mistral 7B) instead of the full sequence. Cost becomes $O(n \times W)$ instead of $O(n^2)$, and the KV cache can be a fixed-size rolling buffer.

How global context is retained: layers are stacked. Token at layer 1 sees W tokens back; at layer 2 it sees tokens that themselves saw W tokens back, so the **receptive field grows by W per layer** – after k layers it covers about k × W tokens (Mistral: 32 layers × 4k ≈ 131k tokens), similar to how stacked CNN layers grow their receptive field. Some models also mix in a few

global-attention layers or global tokens (Longformer, Gemma 2 alternating local/global layers).

Q8. What are the types of positional embeddings (absolute, learned, relative, RoPE, ALiBi)? Absolute vs relative? ★

Answer

- **Absolute sinusoidal** (original Transformer) – fixed sine/cosine vectors added to embeddings.
- **Absolute learned** (BERT, GPT-2) – a trainable embedding per position; cannot go beyond max length (512/1024).
- **Relative position** (Transformer-XL, T5 relative bias) – encode the **distance between tokens** ($i - j$) inside attention instead of absolute positions.
- **RoPE – Rotary Position Embedding** (Llama, Mistral, Qwen, GPT-NeoX) – rotates Q and K vectors by an angle proportional to their position, so the dot product depends only on relative distance.
- **ALiBi** (BLOOM, MPT) – adds a linear penalty to attention scores proportional to distance; good length extrapolation.

Absolute vs relative: absolute embeddings encode "I am at position 5"; relative embeddings encode "this token is 3 positions before me". Relative methods generalize better to longer sequences and capture what really matters (distance between words), while absolute learned embeddings are limited to the trained length.

Q9. How does RoPE help models extend to longer context windows?

Answer

RoPE encodes position as a **rotation**: dimension pairs of Q and K are rotated by angle pos $\times \theta_i$ (different frequency per pair). The attention score $q \cdot k$ then depends only on the **relative distance** between the tokens, not their absolute positions.

For context extension, the rotation frequencies can be **rescaled** so that longer sequences map into the angle range seen during training:

- **Position Interpolation** – compress positions (e.g. scale by 4 to go from 4k → 16k).
- **NTK-aware scaling** and **YaRN** – scale different frequencies differently, keeping local detail.
- Increase the RoPE **base** (e.g. 10,000 → 500,000 in Llama 3).

With a short fine-tune on long documents (or sometimes none), models extend from 4k to 32k–128k+ context.

Q10. What is a Mixture-of-Experts (MoE) model and why is it popular?

Answer

A **Mixture-of-Experts (MoE)** model replaces the single feed-forward network in each Transformer block with **many expert FFNs** plus a **router (gating network)**. For each token, the router picks the **top-k experts** (usually 1–2) and only those are computed.

Example: **Mixtral 8×7B** – 8 experts, 2 active per token → ~47B total parameters but only ~13B **active** per token. (DeepSeek-V3, Qwen-MoE, and reportedly GPT-4 also use MoE.)

Why popular: much larger model capacity (knowledge) at the **compute cost of a much smaller model** → better quality per FLOP, faster training and inference.

Challenges: all experts must still be in memory (high VRAM), load balancing between experts (auxiliary loss), training instability, more complex distributed serving.

Q11. Advantages and disadvantages of the Transformer architecture.

Answer

Advantages: parallel training; captures long-range dependencies; scales to huge models; transfer learning via pre-training; state-of-the-art across NLP, vision, audio and multimodal; flexible (encoder, decoder, encoder–decoder).

Disadvantages:

- **O(n²)** time and memory in sequence length → expensive for long contexts.
- Needs huge data and compute to pre-train; high inference cost.
- Fixed/limited context window.
- No built-in notion of order (needs positional encodings).
- Hard to interpret; can hallucinate; inherits data biases.
- Large memory footprint (weights + KV cache).

Q12. What is a language model? Compare autoregressive vs masked language modelling.

Answer

A **language model** assigns probabilities to sequences of words – it learns P(text) and can predict missing/next words.

- **Autoregressive (causal) LM** – predicts the **next token** from the previous ones: P(x_t | x_1 … x_(t−1)). Unidirectional; natural for **generation**. Examples: GPT, Llama.
- **Masked LM (MLM)** – randomly hides ~15% of tokens and predicts them using **both left and right context**. Bidirectional; great for **understanding** tasks, but cannot generate text naturally. Examples: BERT, RoBERTa.

(Seq2seq LMs like T5 use span corruption: mask spans in the encoder, regenerate them with the decoder.)

Q13. What is BERT? How is it bidirectional? ★ · Asked: NLP interview rounds

Answer

BERT (Bidirectional Encoder Representations from Transformers) – Google, 2018 – is a stack of **Transformer encoder** layers pre-trained on large text (Wikipedia + BooksCorpus) and then fine-tuned for downstream tasks. BERT-base: 12 layers, 768 hidden, 12 heads, 110M parameters; BERT-large: 24 layers, 1024 hidden, 16 heads, 340M.

How it is bidirectional: it uses **un-masked self-attention**, so every token attends to all tokens on **both left and right** at every layer at the same time. This is possible because it is trained with **Masked Language Modelling** – the word to predict is hidden, so seeing both sides is not cheating. (Unlike ELMo, which just concatenated separate left-to-right and right-to-left LSTMs, BERT is "deeply" bidirectional.)

Q14. Which tasks are used for BERT pre-training (MLM and NSP)? Why did RoBERTa drop NSP? ★

Answer

1. **Masked Language Modelling (MLM):** 15% of tokens are selected; of these, 80% are replaced with [MASK], 10% with a random token, 10% left unchanged; the model predicts the original tokens. (The 80/10/10 split reduces the mismatch because [MASK] never appears during fine-tuning.)
2. **Next Sentence Prediction (NSP):** given sentence A and B, predict whether B actually follows A (50% real, 50% random) using the [CLS] output. Intended to help tasks like QA and

NLI.

Why RoBERTa dropped NSP: experiments showed NSP **did not help and sometimes hurt** performance – the task was too easy (random sentences usually differ in topic) and it forced training on short segment pairs. RoBERTa instead trained only with MLM on full-length contiguous text, with dynamic masking, bigger batches, more data and longer training, and outperformed BERT.

Q15. Cased vs uncased BERT — when would you use each?

Answer

- **Uncased** – text is **lowercased and accents are removed** before WordPiece tokenization ("Apple" = "apple"). Smaller effective vocabulary; better when case carries no meaning. Default choice for most tasks (sentiment, classification).
- **Cased** – keeps **upper/lower case and accents**. Use when case matters: **Named Entity Recognition** ("Apple" the company vs "apple" the fruit), POS tagging, acronyms, and for languages where accents matter.

Q16. Which tokenizer does BERT use? What do [PAD], [UNK], [CLS], [SEP] and [MASK] represent?

Answer

BERT uses the **WordPiece** tokenizer (vocabulary 30,522 for bert-base-uncased); sub-word continuations start with "##".

Special tokens:

- **[PAD]** (id 0) – padding so all sequences in a batch have the same length; ignored via the attention mask.
- **[UNK]** (id 100) – unknown token for characters/words not in the vocabulary.
- **[CLS]** (id 101) – added at the **start** of every input; its final hidden state is used as the sequence representation for classification.
- **[SEP]** (id 102) – separates two sentences and marks the end of the input.
- **[MASK]** (id 103) – replaces tokens during MLM pre-training.

Input format: [CLS] sentence A [SEP] sentence B [SEP] (+ token_type_ids 0/1 for segments).

Q17. How do you fine-tune BERT for classification, NER and question answering?

Answer

Add a small task-specific head on top of BERT and train the whole model (small LR, e.g. 2e-5, 2–4 epochs):

- **Classification** (sentiment, intent): take the **[CLS] output** → Dropout → Linear(768 → num_classes) → softmax. `AutoModelForSequenceClassification`.
- **NER / token classification:** take **every token's output** → Linear(768 → num_tags) to predict BIO tags per token; align labels with sub-word tokens (label only the first sub-token, set others to −100). `AutoModelForTokenClassification`.
- **Extractive QA (SQuAD):** input "[CLS] question [SEP] context [SEP]"; two vectors predict the **start and end positions** of the answer span in the context. `AutoModelForQuestionAnswering`.

```
from transformers import AutoModelForSequenceClassification
model = AutoModelForSequenceClassification.from_pretrained("bert-base-uncased", num_labels=3)
```

Q18. NLP case study: you must classify customer complaints into 10 categories. Walk through your approach from baseline to the final model, including class imbalance. · **Asked: NLP interview rounds**

Answer

1. **Understand the data:** volume, labels quality, class distribution, text length, languages (Hinglish?).
2. **Baseline:** light cleaning → **TF-IDF (1–2 grams) + Logistic Regression / Linear SVM**. Fast, strong, explainable; gives a benchmark.
3. **Metrics:** macro-F1 and per-class precision/recall (accuracy is misleading with imbalance); confusion matrix to see which categories get mixed up.
4. **Class imbalance:** class weights (`class_weight="balanced"`), oversampling minority classes or augmentation (back-translation, paraphrasing with an LLM), focal loss, more labelled data for rare classes, threshold tuning.
5. **Stronger model:** fine-tune **DistilBERT/BERT/MuRIL** (for Indian languages) with a classification head; usually +5–15 F1 points over TF-IDF.
6. **Low-data alternative:** zero/few-shot classification with an LLM, or use the LLM to label data, then train a small model.
7. **Error analysis**, then deploy with monitoring (drift in categories, low-confidence routing to humans).