1. What is language modeling?

The main concept of language modelling is to estimate how well a model could predict the next letter or word of a text or sentence considering that the past N letters or words are known.

2. What is self-supervised pertaining?

Self-supervised pretraining is an approach to pre-training which uses self-supervised learning. In this approach, a neural network is trained using the supervision signals generated by itself, rather than those provided by humans. Pre-training is a process where before being trained for a particular task, the neural network is optimized as a result, it does not need to train a deep, complex neural network from scratch on tasks with limited labelled data. In self-supervised pre-training does not need the initial model for annotating the data. Instead, all the supervision signals are created from the text, and the entire model is trained from scratch.

3. Why is pretraining more hardware-efficient for Transformer- or attention-based models compared to RNN-based models?

Transformers allow parallelization across entire sequences during training due to their self-attention mechanism. RNNs require sequential processing (one token at a time), making them inefficient on modern parallel hardware like GPUs and TPUs. This makes Transformers significantly more efficient to pretrain at scale.

4. What is the difference between encoder-only and decoder-only models, and why are decoder-only models more popular?

Encoder-only models are well suited for text classification system as it is good at understanding tasks. (e.g. BERT) The decoder-only model predicts the distribution of tokens at a position given its preceding tokens, and the output is taken with maximum probability. The decoder-only model generates text auto-regressively and are well suited for generative tasks. (e.g. GPT) The decoder-only models are more popular due to its versatility and its ability to apply the model to generating tasks. It can support zero-shot/few-shot/instruction-following tasks better.

5. Suppose the vocabulary consists of only three words: *Apple*, *Banana*, and *Cherry*. During decoder-only pretraining, the model outputs the probability distribution $Pr_\theta(\cdot \mid x_0, \dots., x_i) = (0.1, 0.7, 0.2)$. If the correct next word is *Cherry*, represented by the one-hot vector $(0, 0, 1)$, what is the value of the log cross-entropy loss? What is the loss value if the correct next word is *Banana* instead?

$$L(Cherry, (0.1, 0.7, 0.2)) = -(0 \times \ln 0.1 + 0 \times \ln 0.7 + 1 \times \ln 0.2) \approx 1.609$$

$$L(Banana, (0.1, 0.7, 0.2)) = -(0 \times \ln 0.1 + 1 \times \ln 0.7 + 0 \times \ln 0.2) \approx 0.357$$

6. What are zero-shot learning, few-shot learning, and in-context learning?

Zero-shot learning: It occurs when a model which has already been fine-tuned can handle new tasks without fine-tuning or training it.

Few-shot learning: This learning is achieved through in-context learning however only a small number of dataset/samples would demonstrate how an input corresponds to an output. They are known as demonstrations.

In-context learning: In this case when a model is prompted it would provide it with new information to the context such as demonstrations of problem solving. The LLM will learn from the context how to solve the problem. By providing examples/context in the input prompt the model will have the ability to learn on the fly.

7. Why is fine-tuning necessary? What is instruction fine-tuning?

Using a pre-trained model which can be applied to various NLP tasks, with the help of fine-tuning the model which has an overall general knowledge about the task could become a more specialised model. Essentially it is a mechanism to activate this knowledge for applying it to new tasks. In instruction fine-tuning, the model parameters are fine-tuned using instruction-following data. In instruction fine-tuning, the input includes instructions, system information and any other user-provided information to assist the model to arrive to the right outcome. System information refers to sequence of tokens added at the beginning of an input to guide the behaviour of an LLM.

8. What is tokenization? What is a word embedding layer?

Tokenization is the process of creating tokens which are the basic units of text. Work embedding layer is a way to map words/tokens into low-dimensional real-valued vectors in a continuous space, making it possible for the model to compute the meaning of the words.

9. What is position embedding? What kind of position embedding method is used in Llama models?

Position embedding is adding order to word embeddings because work embedding provides semantic meaning but lack order awareness. With transformers they process the inputs in parallel, so it does not consider the positional information. Thus, the positional embedding compensates for that weakness. In Llama models it uses rotary positional embedding (RoPE) which encodes position through sinusoidal transformations applied in the attention mechanisms.

10. What is the difference between multi-head attention and grouped-query attention? Which type of attention mechanism is used in Llama models?

In multi-head attention (MHA) the attention module repeats its computations multiple times in parallel. Each of these is called an attention head. The module is split its Query, Key and Value parameters N-ways and passes each split independently through a separate Head. All these similar attention calculations are then combined to produce a final Attention score.

Grouped-query attention (GQA) is a technique used in LLM to speed up inference time by grouping queries together and computing their attention collectively. It reduces the compute/memory by grouping queries to share key-value projections.

Llama models use grouped-query attention (GQA) as their attention mechanism in their models.

11. What kind of activation function is used in Llama models?

In Llama models it employs SwiGLU (Swish-Gated Linear Unit) activation function. It is a variant of GLU (Gated Linear Unit).

12. What is layer normalization? What is the difference between layer normalization and batch normalization?

Layer normalization, all neurons in a particular layer effectively have the same distribution across all features for a given input. It normalizes the activation across the features of each individual training instance. Whereas in batch normalization, it normalizes the activation across a mini batch. As batch normalization is dependent on batch size, it is not effective for small batch sizes whereas layer normalization is independent of batch size so it can be applied where the batch size is small. Layer normalization is one of the main features in Transformer architectures and it is being using in Llama models.

13. What is the auto-regressive generation process, and how is a decoding strategy used during text generation?

Auto-regressive generation process allows models to generate content (e.g. text) based on the limited information (e.g. tokens) provided to the model. Essentially it produces token based on previously generated tokens as inputs.

$$x_i = \mathrm{argmax}_{\mathrm{x}} \Pr(x_i | x_0, \ldots., x_{i-1})$$

Where x_i is the predicted token.

Decoding Strategies dictate how a language model selects the next token in a sequence after predicting probabilities for all possible tokens. The following the different strategies used:

Deterministic Approach:

- Greedy Search: It is a deterministic approach where is selects the most probable token at each step as the next token in the sequence.
- Beam Search: It systematically extends these sequences at each step by evaluating all possible next tokens and calculating the probabilities of resulting sequences. The sequences with the highest cumulative probabilities are kept for further expansion in subsequent iterations. This process continues until the sequences reach a maximum length or generate an end token. Finally the sequence with the highest total probability among the beams is selected as the final output.

Stochastic Approach:

- Sampling: Randomly samples from the distribution.
- Top-k: In top-k sampling it refines the token selection process by restricting the choice to the top-k most likely candidates. From the pool of k samples, it chooses the next token based on its probability.
- Top-p (nucleus) sampling: The pool of potential next tokens is determined by the cumulative probability of the most likely tokens. When the threshold is set to p, the model includes enough of the top probable tokens so that their cumulative probability meets or exceeds this threshold.