Phase 1

NIVEDITHA MADEGOWDA AND GAYATRI SRIDHAR

Background Knowledge : In this phase, you will review the shortened version of the LLM Foundations paper provided in the project repository. After reading through the paper, please answer the questions listed below. You are welcome to use external resources like Google or ChatGPT to help with your understanding, but you are responsible for ensuring the accuracy and completeness of your responses.

1. What is language modeling?

Answer: Language modeling is a task in natural language processing (NLP) that involves predicting the probability distribution of a sequence of words in a language. That is "how well can the next letter of a text be predicted when the preceding N letters are known." These days it is used in text generation, sentiment analysis, speech recognition etc

2. What is self-supervised pertaining?

Answer: The self-supervised model does not depend on the labelled data for training in the initial phase. Instead the model learns useful representations (features) from the raw, unlabeled data. An entire model is trained from scratch.

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

Answer: Parallelization: Transformers use self-attention mechanisms that process all tokens in a sequence in parallel during both training and inference. Since hardware accelerators are efficient at parallel processing, they allow for better utilisation of resources.
RNNs process tokens one at a time, since the output of each time step depends on the previous time step's output (the recurrent part) thus preventing parallelization.

Memory Access: Transformers have more efficient memory access patterns due to matrix operations being highly optimized on accelerators, thereby allowing faster training.

Scalability: Transformers scale better with long sequences since the attention mechanism operates on the entire sequence at once. They converge faster as they can process entire sequence at once. In RNNs this gets difficult as they need to be processed in order.

Training Speed: The self-attention mechanism computes relationships between all pairs of tokens in a single step and can model long range dependency. But in RNNs, they capture sequential dependencies and are inefficient in capturing long range dependencies.

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

Answer: Encoder-only models do not generate text. They capture the relationships and context within the input data. The goal is to understand the text by utilising self-attention mechanism where input text is

processed through multiple layers of the encoder. It produces a fixed-size representation of the input, which can then be used for tasks like text classification, sentiment analysis etc.

Decoder-only models are used to generate text using decoders. They generate one token at a time based on the input and previously generated tokens. These models use masked self-attention, which allows the model to attend to all previous tokens in the sequence when generating the next token. They are well-suited for tasks involving text generation, such as language modeling, text completion, and creative writing

Decoder-only models are very popular because they can be used in variety of tasks such as text generation, code generation, and creative writing. Encoder-only models will be useful mostly in business use cases compared to decoder-only models. Their ability to perform well with large-scale pretraining, along with their autoregressive nature, makes them highly versatile and efficient for many modern NLP applications.

5. Suppose the vocabulary consists of only three words: Apple, Banana, and Cherry. During decoder-only pretraining, the model outputs the probability distribution Prθ(· | x0, . . . , xi) = (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?

$$\text{LogCrossEntropy}(p^{\theta}_{i+1}, p^{\text{gold}}_{i+1}) = -\sum_{j} p^{\text{gold},j}_{i+1} \cdot \log(p^{\theta}_{i+1}, j)$$

Answer:

So logCross Entropy for **cherry** =-[0. log(0.1) + 0 . log(0.7) + 1 . log(0.2)] = 1.6094

If correct next word is **banana:**

- [0. log(0.1) + 1 . log(0.7) + 0 . log(0.2)] = 0.3567

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

Answer: zero-shot learning refers to a model where it performs a task without any example prompt and only pre-trained knowledge.

Few-shot learning is where few examples 2 or 3 are given about how a task should be done and then model generates the data according to these examples given

In-context learning: This used both examples as well as pretrained data to generate text. It learns by seeing the prompt context as well as examples.

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

Answer: In general language models are trained on vast amounts of data to understand broad language patterns. These might not be able to generalise well on one particular topic. In order to make the model well versed for one particular domain, fine-tuning is necessary. It has adds new capabilities and cut down on unnecessary tasks.

Instruction fine-tuning is a specialized type of fine-tuning where the pretrained language model is further trained on datasets containing explicit instructions and corresponding desired outputs. It teaches

the model based on the given instructions which is useful in tasks which makes it effective for tasks involving following a command.

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

Answer: Tokenization involves splitting text into smaller units called tokens, which are meaningful pieces that a model can process. Tokens can be words, subwords, characters, or even punctuation marks, depending on the tokenization strategy used.

A **word embedding layer** is a neural network layer that converts the token indices into high-dimensional, continuous numerical vectors called **word embeddings**. These embeddings capture semantic and contextual meaning of words in a vector form that neural networks can easily understand and learn from. This approach allows models to better understand the context and meaning of words, leading to improved performance in various natural language processing tasks.

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

Answer: In transformer based language models, positional embedding encodes the position of each word or token in a sequence as they process the words in parallel and do not store the order data. Encoding the positions helps in understanding the sequence/context or order of the words. There are 2 types - absolute position embedding(fixed embedding vector) and relative positional embedding(encodes relative distances)

LLaMA models use Rotary Positional Embedding (RoPE), a type of relative positional embedding.RoPE encodes positional information by applying a **rotary transformation** to token embeddings. Each embedding dimension pair is rotated based on token position, using predefined sinusoidal functions. It provides stable performance, strong generalization capabilities, and good handling of very long contexts.

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

Answer: Multihead attention - Each attention head has its own separate projection matrices for queries (**Q**), keys (**K**), and values (**V**). They require significant memory and computational resources.

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

Grouped Query Attention : Multiple attention heads share keys and values, but have separate queries. There are fewer parameters and computational complexity is reduced as they share keys and values

$$Q_{\text{head}_i}(\text{unique}), \quad K, V(\text{shared among groups of heads})$$

Llama models use Group Query Attention mechanism.this significantly improves computational efficiency, reduces memory usage, and scales effectively to larger model sizes without sacrificing performance.

11. What kind of activation function is used in Llama models?
Answer: LLaMA models use the SiLU (Sigmoid Linear Unit) activation function, also known as the Swish activation function.

$$\text{SiLU}(x) = x \cdot \sigma(x) = x \cdot \frac{1}{1 + e^{-x}}$$ where σ(x)represents the sigmoid function.
This provides a smooth, continuous gradient, helping models converge faster during training.

12. What is layer normalization? What is the difference between layer normalization and batch normalization?
Answer:Layer Normalization normalizes the activations across features within each individual input sample, independently from other samples in the batch**. T**hey are usually used in transformers.

$$\text{LayerNorm}(x) = \gamma \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta$$

Where:

- x is the input vector to a layer (one individual training example).
- μ and σ^2 are mean and variance computed over the features of a single example.
- γ and β are learnable parameters for scaling and shifting.
- ϵ is a small constant for numerical stability.

Batch normalization: In contrast, batch normalization normalizes across the entire batch dimension for each feature independently. Depends heavily on batch size; small batch sizes can degrade performance. They are commonly used in CNNs

$$\text{BatchNorm}(x) = \gamma \frac{x - \mu_{\text{batch}}}{\sqrt{\sigma^2_{\text{batch}} + \epsilon}} + \beta$$

Where:

- μ_{batch} and σ^2_{batch} are computed across a mini-batch of data points, for each feature separately.
- γ and β are learnable scaling/shifting parameters.

13. What is the auto-regressive generation process, and how is a decoding strategy used during text generation?
Answer: The **auto-regressive generation** process refers to generating output tokens sequentially, one token at a time, where each new token is conditioned on the previously generated tokens. Formally, given a sequence of tokens: xo, x1, x2…xm-1,
the model predicts the next token xm based on the past tokens: xm = P(xm | x0, x1, x2,....xm-1) this continues until a stopping criterion is met.

A **decoding strategy** is used during auto-regressive text generation to select the next token from the predicted probability distribution. Various strategies are:

Greedy Decoding:Selects the token with the highest probability at every step.
Beam Search: Maintains multiple candidate sequences (the "beam") at each step, selecting sequences with the highest combined probabilities.
Sampling Methods (e.g., Top-k, Top-p Sampling): Select tokens probabilistically from the top-k or top-p most likely tokens, introducing randomness.

- **Top-k Sampling**: Samples from the top-k most probable tokens.
- **Top-p (Nucleus) Sampling**: Samples from the smallest possible set of tokens whose cumulative probability exceeds a threshold ppp.