EE 508 Phase 1

What is Language Modeling?

Language modeling involves predicting the probability distribution of words in a sequence. It aims to estimate the likelihood of the next word (or token) given the previous words. Formally, language modeling is about computing $P(x_i | x_0, x_1, \dots, x_{i-1})$.

What is self-supervised pretraining?

Self-supervised pretraining is a method where a model learns from unlabeled data by generating its own training signals, rather than relying on human-annotated labels. In NLP, this typically involves creating tasks from raw text—such as predicting a masked word based on its context—allowing the model to train from scratch without needing any initial labeled data or seed models. Unlike traditional self-training, which begins with a small labeled dataset to create pseudo labels, self-supervised learning extracts all supervision directly from the data itself. This approach has become a cornerstone of modern NLP, enabling large-scale training that powers many tasks in language understanding, generation, and reasoning.

Why is pre training more hardware-efficient for Transformer- or attention-based models compared to RNN-based models?

Pre-training is more hardware-efficient for Transformer-based models because of how they handle input sequences. Transformers rely on self-attention mechanisms that allow them to process all tokens in a sequence simultaneously, rather than step-by-step. This parallelism makes it possible to batch large amounts of data and fully utilize modern GPU or TPU hardware, which are optimized for matrix operations and parallel computation.

In contrast, RNN-based models process sequences sequentially — each token's output depends on the computation of the previous token. This dependency chain prevents efficient parallelization, especially over long sequences, creating a bottleneck in training speed and scalability.

As a result, Transformer models scale better, train faster, and make more effective use of compute resources during pre-training. This efficiency is one of the key reasons why they've largely replaced RNNs in large-scale NLP systems.

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

Encoder-only models, like BERT, are designed to understand text. They take in the full input at once and are trained to predict masked words using the surrounding context. This makes them great for tasks like classification, sentiment analysis, and question answering where deep understanding of input is key.

Decoder-only models, like GPT, are built for generation. They process text left to right, predicting one word at a time. This setup makes them naturally suited for open-ended tasks like writing, summarizing, and answering questions.

In recent years, decoder-only models have become more popular — not because encoder-only models aren't useful, but because decoder-only ones are more flexible. They support **in-context learning**, meaning they can perform new tasks just by being shown examples in a prompt, without needing to be retrained. They're also simpler to pretrain (just next-word prediction), and this single approach generalizes well across many tasks. Combined with strong performance and momentum from models like GPT, decoder-only models have become the go-to choice for many applications.

Suppose the vocabulary consists of only three words: Apple, Banana, and Cherry. During decoder-only pretraining, the model outputs the probability distribution $P(. \mid x_0, \ldots, x_i) = (0.1, 0.7, 0.2)$**. If the correct next word is Cherry, represented by 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?**

$P(. \mid x_0, \ldots, x_i) = (0.1, 0.7, 0.2)$

- Log cross-entropy loss if correct next word is Cherry: $-log(0.2) = 1.609$
- Log cross-entropy loss if correct next word is Banana: $-log(0.7) = 0.357$

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

Zero-shot learning: Model performs a new task without explicit training examples, typically via prompts or instructions.

Few-shot learning: Model generalizes from a small number (e.g., 1-5) of examples provided in the input prompt.

In-context learning: Learning entirely during inference by conditioning on examples provided in the input (prompt), without further training or parameter updates.

What is tokenization? What is a word embedding layer?
Tokenization: Converting raw text into a sequence of tokens, such as words, subwords, or characters.

Word embedding layer: Maps tokens to continuous vector representations that capture semantic meaning and syntactic context.

What is position embedding? What kind is used in LLaMA models?

Transformers don't inherently understand the order of tokens, so position embeddings inject information about token positions into the model. LLaMA models use Rotary Positional Embeddings (RoPE), which rotate the query and key vectors in multi-dimensional space using sinusoidal functions. This allows the model to encode relative positions naturally, making it better at generalizing to longer sequences and maintaining contextual alignment across tokens.

What is the difference between multi-head attention and grouped-query attention? Which type of attention mechanism is used in LLaMA?
Multi-head attention runs several attention mechanisms in parallel, each with its own query, key, and value projections, capturing different aspects of token relationships.
Grouped-query attention (GQA), used in LLaMA, reduces memory and compute by sharing key and value projections across multiple query heads. This maintains strong performance while being more resource-efficient — a big advantage for scaling up large models.

What kind of activation function is used in Llama models?
Llama models use the **SiLU (Sigmoid Linear Unit)** activation function, also known as the Swish function: $SiLU(x) = x \cdot \sigma(x)$

This function smoothly blends linear and nonlinear behaviors, allowing for better gradient flow and model expressiveness than traditional ReLU.

What is layer normalization? WHat is the difference between layer normalization and batch normalization?
Layer normalization normalizes the hidden state of each token individually across its features, making it independent of other examples in the batch — ideal for sequential data like text. Batch normalization instead normalizes across the batch, which isn't as effective for NLP. LLaMA, like most Transformer models, uses layer normalization for its stability and suitability in sequence modeling.

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

Autoregressive generation predicts tokens sequentially, each token prediction conditioned on previous tokens.

Decoding strategies include:

- Greedy decoding: Picks highest probability token at each step (fast, but suboptimal).
- Sampling methods (e.g., top-k, nucleus sampling): Introduces randomness, better diversity.
- Beam search: Searches multiple hypotheses simultaneously, often used in structured generation tasks like translation, balancing quality and computational cost.