**Project: IMDB Movie Sentiment Classification with Transformers**

**Part 1 – Self-Attention Fundamentals**

* **Self-Attention Mechanism:** Computes weighted averages between input tokens so each output reflects relationships across the sequence.
* **Basic Implementation:** Efficient matrix operations (torch.bmm) for weighted averages and dot-product attention; softmax normalizes attention scores.
* **Maximum Sequence Length:** Input text tokenized and padded/truncated to a pre-defined length (e.g., 512 for BERT).

**Part 2 – Transformer Model Architecture**

* **Multi-Head Self-Attention:** Input vectors are projected into separate Query, Key, and Value spaces for each head. Each head processes a different view, capturing diverse dependencies.
* **Custom PyTorch Module:** Combines attention (SelfAttention class), feed-forward layers, layer normalization, and residual connections (TransformerBlock class).
* **Positional Embeddings:** Required because Transformers’ architecture treats input as a set—positional embeddings encode sequence order.

**Part 3 – Transfer Learning and IMDB Dataset**

* **Data Loading:** Uses Hugging Face datasets and transformers libraries to prepare IMDB movie review dataset; text is tokenized and labeled for classification.
* **Backbone Feature Extractor:** Pre-trained DistilBERT model extracts contextual token representations from input text.
* **Classifier Head:** Custom Transformer layers and a linear output layer map representations to class logits (positive/negative review).

**Part 4 – Training and Evaluation Pipeline**

* **Frozen Pretrained Layers:** Only custom Transformer and classifier layers are trained; BERT encoder is frozen for efficiency and stability.
* **Training Loop:** Functions modularized for training, validation, and prediction; metrics tracked per epoch (accuracy, loss).
* **Optimizer & Scheduler:** Adam optimizer, with warmup strategy for adaptive learning rate.

**Part 5 – Model Analysis & Attention Visualization**

* **Attention Weight Visualization:** Utility functions plot which tokens are focused on by each attention head; reveals model’s interpretative behavior.
* **Embedding Inspection:** Cosine similarity among learned word and positional embeddings highlights semantic clustering and sequence-based patterns.

**Part 6 – Key Insights**

* **Multi-Head Attention:** Enables richer modeling by letting different heads focus on distinct aspects and relationships in the input.
* **Contextual Representation:** Transformers process embeddings in context, unlike static word vectors, enabling nuanced language understanding.
* **Positional Embeddings:** Essential for encoding sequence order, which would be lost otherwise.
* **Transfer Learning:** Using pre-trained models greatly boosts downstream task performance with fewer labeled examples.

**Interview Preparation: Key Concepts**

**Self-Attention & Transformers**

* **What is self-attention, and how does it work?**  
  Self-attention computes weights for each token in a sequence, letting the model dynamically focus on relevant information from other tokens by taking a weighted average of their representations.
* **Why do Transformers use positional embeddings? What happens without them?**  
  Transformers process inputs in parallel and lack inherent sequence order; positional embeddings ensure the model knows token locations. Without them, the model treats input as a set, losing all order-dependent meaning.
* **How is multi-head self-attention different from single-head?**  
  Different heads have independent sets of learned parameters for query, key, and value mappings. This lets each head focus on distinct aspects or relationships, capturing richer and more diverse dependencies.

**Transfer Learning / Pre-trained Models**

* **Why use a pre-trained model like BERT for feature extraction?**  
  Pre-trained models capture general language features from large corpora, enabling robust and transferable representations. This boosts performance and makes training much more efficient for downstream tasks like classification.
* **What is frozen fine-tuning, and why is it helpful?**  
  In frozen fine-tuning, the pre-trained backbone’s weights remain unchanged. Training only the custom layers ensures stability and efficiency, especially with limited data or compute.

**Model Training & Evaluation**

* **Why use modular training loops and metrics functions?**  
  Modular functions promote code reuse, avoid bugs, and make it easier to track meaningful metrics (e.g., accuracy, loss) during both training and validation.
* **What challenges arise with long sequence inputs?**  
  Transformers’ self-attention scales quadratically with input length, so longer sequences demand much more compute and memory. Attention can dilute, and capturing long-range dependencies may become harder for very long texts.

**Embeddings & Context**

* **Why aren’t word embeddings alone enough for classification?**  
  Fixed word embeddings don’t capture context—words with multiple meanings stay ambiguous. Transformers build contextualized representations, adapting meaning based on surrounding words.
* **What does inspecting learned embeddings tell us?**  
  Semantic clustering among tokens (e.g., synonyms have similar vectors); positional embedding similarity reflects token sequence order. Special tokens (like [CLS]) show distinctive patterns due to their unique roles.