1. pre-training, adaptation tuning, utilization, and capacity evaluation

   a. pre-training (how to pretrain a capable LLM)

   b. adaptation (how to effectively adapt pre-trained LLMs for better use)

   c. utilization (how to use LLMs for solving various downstream tasks)

   d. capability evaluation (how to evaluate the abilities of LLMs and existing empirical findings)

2. The term large language models (LLM) represents for the PLMs of significant size (containing tens or hundreds of billions of parameters)

   1. Large Language Models (LLMs) have emergent abilities that were not present in smaller pre-trained language models (PLMs). These abilities significantly improve their performance on complex tasks.
   2. LLMs are set to change the manner in which humans develop and interact with AI algorithms. This impact is considerably different from the impact of smaller PLMs.
   3. Access to LLMs is typically through a prompting interface, such as the GPT-4 API, which requires humans to learn how to effectively communicate with LLMs to maximize their potential.
   4. The evolution of LLMs has blurred the lines between research and engineering because their development relies on practical skills in handling large-scale datasets and managing distributed, parallel training processes.

3. LLMs can be categorized into three major types, namely encoder-decoder, causal decoder, and prefix decoder.

   a. Encoder-decoder Architecture

   The initial Transformer model is built on the encoder-decoder architecture, containing two stacks of Transformer blocks as the encoder and decoder.

   There are only a small number of LLMs that are built based on the encoder-decoder architecture, such as Flan-T5.

   b. Causal Decoder Architecture

   The causal decoder architecture incorporates the unidirectional attention mask, to guarantee that each input token can only attend to the past tokens and itself.

   The causal decoders have been widely adopted as the architecture of LLMs by various existing LLMs, such as OPT, BLOOM, and Gopher.

   c. Prefix Decoder Architecture

   The prefix decoder architecture (non-causal decoder) revises the masking mechanism of causal decoders, to enable performing bidirectional attention over the prefix tokens and unidirectional attention only on generated tokens.

   Examples are GLM-130B and U-PaLM.

4. LM aims to model the generative likelihood of word sequences, so as to predict the probabilities of future (or missing) tokens.

a. Causal Language Models (CLM) generate text in a forward direction, learning to predict each subsequent token based on the tokens that came before it.

b. Masked Language Models (MLM) can take into account the entire context, learning to understand and predict tokens based on all surrounding text, which allows them to better understand the meaning of words in context.

5. Text classification task involves assigning predefined categories (or labels) to text. For example, an email might be classified as "spam" or "not spam," or a movie review might be categorized as expressing a "positive" or "negative" sentiment. The task is a fundamental one in the field of natural language processing (NLP), as it forms the basis for many applications that need to understand and organize text at scale.

   To perform text classification, one would typically use a model that has been trained to understand language patterns. The types of models can include:

   a. Traditional Machine Learning Models: Such as Naive Bayes, Support Vector Machines (SVM), or Random Forests, which often rely on bag-of-words or TF-IDF (Term Frequency-Inverse Document Frequency) features.

   b. Neural Network Models: Such as Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs), including their variants like Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) networks.

   c. Transformer-based Models: Modern approaches often utilize transformer-based architectures like BERT (Bidirectional Encoder Representations from Transformers), RoBERTa, or GPT (Generative Pre-trained Transformer) models.

   For most modern applications, transformer-based models are preferred due to their superior ability to capture complex language patterns and context. These models are pre-trained on large corpora of text in a self-supervised fashion and can then be fine-tuned on a specific text classification task.

6. Text Summarization methods are grouped into two main categories: Extractive and Abstractive. Extractive Text Summarization, where the model “extracts” the most important sentences from the original text, is the more traditional method. Extractive Text Summarization does not alter the original language used in the text.

   Here are some kinds of models can be used in summarization task.

   a. TextRank algorithm. Google uses an algorithm called PageRank in order to rank web pages in their search engine results. TextRank implements PageRank in a specialized way for Text Summarization, where the highest “ranking” sentences in the text are the ones that describe it the best.

   b. LexRank which works similarly to TextRank, utilizing the singular value decomposition of a word-sentence matrix.

   c. T5. It is a Transformer based architecture that uses a text-to-text approach.

7. a. Adam and AdamW are widely used optimizers for training large language models (LLMs) due to several reasons:

   1. **Adaptive Learning Rates**: Both Adam and AdamW use adaptive learning rates for each parameter. This means they compute individual learning rates for different parameters from estimates of first and second moments of the gradients. This feature is particularly

useful for complex models like LLMs, where different parameters may require different learning strategies.

2. **Efficient with Large Datasets and Parameters**: They are efficient for large-scale optimization problems common in LLMs, where datasets and model sizes are substantial.
3. **Balancing Exploration and Exploitation**: These optimizers strike a balance between exploration of the parameter space (which SGD with momentum does) and exploitation (which is typical of algorithms that adapt the learning rate based on the variance).
4. **Robustness**: Adam and AdamW are more robust to hyperparameter settings, especially the learning rate, which makes them a preferred choice when fine-tuning LLMs.

Relationship: AdamW is a variant of Adam that modifies the weight decay component, decoupling it from the gradient updates. This can lead to better training dynamics and ultimately better performance on some tasks.

b. Both Adam and AdamW introduce extra overhead in terms of memory usage:

1. Adam optimizer stores:

   First moment estimates (the mean of the gradients), which is an array of size N.

   Second moment estimates (the uncentered variance of the gradients), which is another array of size N.

   Additionally, for each parameter, Adam keeps track of a time step counter to compute the bias-corrected first and second moment estimates.

2. AdamW optimizer stores the same information as Adam but handles the weight decay component differently during the parameter update step.

Thus, both optimizers require storage for at least 3N parameters (the original parameters, the first moment, and the second moment), not including the additional memory for the time step counter. This means that the memory requirements for Adam and AdamW are typically more than double that of simpler algorithms like stochastic gradient descent (SGD) which only needs to store the gradients and the parameters. However, despite the extra memory overhead, the benefits they provide for the convergence and performance of training large models usually outweigh the cost, particularly when using distributed training across multiple GPUs or TPUs where memory resources are abundant.

8. a. A learning rate scheduler is a strategy or a method used during the training of neural networks to change the learning rate during training. The learning rate is a critical hyperparameter that can influence model performance and convergence rates. If it's too high, the training may not converge; if it's too low, training may take too long or get stuck in a local minimum.

   b. **Torch.optim.lr_scheduler.CosineAnnealingLR** is one such learning rate scheduler provided by PyTorch. It implements a cosine annealing schedule, which is characterized by the following behavior:

   Initial Phase: The learning rate starts from a specified initial learning rate (typically higher).

   Middle Phase: As training progresses, the learning rate decreases following a cosine curve from its initial value down to a minimum value. This part of the curve is like the first part of the cosine function, starting at the top of the wave and ending at the bottom.

End Phase: The learning rate then increases again following the curve, but this is often not used in practice since the idea is to decrease the learning rate to fine-tune the model's weights.

The CosineAnnealingLR scheduler can be visualized as a smooth curve resembling the shape of a cosine wave. The purpose of this is to make large adjustments to the learning rate at the beginning of training when we are far from the optimal weights and smaller adjustments as we get closer to the optimal weights. This allows for rapid learning at first but then fine-tuning as the model approaches convergence.

9. Tokenization is the process of converting a sequence of characters into a sequence of tokens. In the context of natural language processing (NLP), tokens are typically words, numbers, or punctuation marks. This process is a fundamental step in many NLP tasks because it structures the input text into a form that can be analyzed and used by algorithms.

   Here are a few different tokenization methods:

   1. **Whitespace Tokenization**: The simplest form of tokenization that splits text on whitespace. It is a naive approach and often does not suffice for complex text structures.
   2. **Dictionary-based Tokenization**: Involves using a dictionary or a lexicon to identify tokens, especially useful for languages where whitespace is not used to demarcate words, like Chinese or Japanese.
   3. **Rule-based Tokenization**: Uses a set of defined rules and regular expressions to identify tokens. This may involve looking for patterns in the text to split it into tokens, such as punctuation or capitalization.
   4. **Subword Tokenization**: Breaks words into smaller units (subwords or characters), which can be beneficial for handling unknown words, morphologically rich languages, or agglutinative languages where words are constructed by combining multiple morphemes.
      - **Byte Pair Encoding (BPE)**: Starts with a large corpus of text and iteratively merges the most frequent pair of bytes (or characters) to create common subword units.
      - **WordPiece**: Similar to BPE, but it merges the pair of tokens that minimizes the likelihood of the training data given the model.
   5. **Morphological Tokenization**: Breaks down words into morphemes, which are the smallest grammatical units in a language, such as stems, root words, prefixes, and suffixes.

   The LLaMA tokenizer is **a BPE model based on sentencepiece**. One quirk of sentencepiece is that when decoding a sequence, if the first token is the start of the word (e.g. “Banana”), the tokenizer does not prepend the prefix space to the string.