Basic Gen – Ai LLM qna

1. What are parameters means such as 7b, 13b, etc.? What is precision?
2. Explain 1-bit, 2-bit, 4-bit, 8-bit, 16-bit quantization.
3. What gets saved as a model, and how is it stored?
4. What is model inference, and how is it done?
5. What are the basic tasks LLMs do, such as NER, token classification, etc.?
6. Why do we classify tokens?
7. What are different types of GPUs, such as consumer-grade or Ampere architecture?
8. What are base models and instruction models?
9. Explain various types of LLM model quantization and saving formats, such as ggml, gguf, gptq, awq, etc.
10. Calculation to check how much memory a model will need based on parameters.
11. Teach me all the basic tasks and how they work:
   - Text Classification
   - Token Classification
   - Table Question Answering
   - Question Answering
   - Zero-Shot Classification
   - Translation
   - Summarization
   - Feature Extraction
   - Text Generation
   - Text2Text Generation
   - Fill-Mask
   - Sentence Similarity

1. Parameters and Precision:

Parameters refer to the trainable weights and biases in a neural network model, including LLMs (Large Language Models). The number of parameters (e.g., 7b, 13b) represents the total number of these trainable values in the model. The more parameters a model has, the more complex patterns it can potentially learn, but also the more data and computational resources are required for training.

Precision refers to the numerical precision or the number of bits used to represent each parameter value. Common precisions include 32-bit (float32), 16-bit (float16), and lower precisions like 8-bit, 4-bit, or even 1-bit. Lower precision can lead to faster computation and smaller model sizes but may result in some loss of accuracy.

2. Quantization:

Quantization is the process of reducing the precision of model parameters from a higher precision (e.g., 32-bit float) to a lower precision (e.g., 8-bit, 4-bit, or even 1-bit) to reduce the model size and improve inference speed. The different quantization levels are:

- 1-bit quantization: Each parameter is represented by a single bit (0 or 1), leading to the smallest model size but potentially significant accuracy loss.
- 2-bit quantization: Each parameter is represented by 2 bits, allowing for 4 possible values.
- 4-bit quantization: Each parameter is represented by 4 bits, allowing for 16 possible values.

- 8-bit quantization: Each parameter is represented by 8 bits, allowing for 256 possible values. This is a common quantization level that balances model size and accuracy.
- 16-bit quantization: Each parameter is represented by 16 bits, providing higher precision than 8-bit but still lower than the original 32-bit float.

Model size: Lower precision means fewer bits are required to represent each parameter, resulting in a smaller overall model size, which can be beneficial for deployment on resource-constrained devices or edge computing scenarios.
- Computational efficiency: Lower precision calculations can be faster and more energy-efficient, especially on hardware accelerators optimized for lower-precision operations.
- Accuracy: In general, lower precision can lead to some loss of accuracy or performance compared to higher precision representations. However, techniques like quantization-aware training and other quantization methods aim to minimize this accuracy degradation.
- Memory requirements: Lower precision models require less memory during inference, which can be advantageous for scenarios with limited memory resources.

3. Model Storage:

LLMs are typically stored as sets of parameters (weights and biases) in various file formats. These formats can be proprietary or open-source, and they may include additional metadata or configurations.

The actual parameters are usually stored as arrays of numerical values, representing the trainable weights and biases of the neural network. These arrays can be compressed or quantized to reduce the overall model size.

4. Model Inference:

Model inference refers to the process of using a trained model to make predictions or generate outputs based on new input data. In the context of LLMs, inference involves feeding input text (or other modalities like images) into the model and obtaining the model's output, such as generated text, classification labels, or other task-specific outputs.

Inference is typically performed on hardware accelerators like GPUs or specialized AI chips to achieve faster computation times. The model parameters are loaded into the accelerator's memory, and the input data is processed through the model's layers to produce the desired output.

5. Basic LLM Tasks:

Large Language Models (LLMs) are capable of performing various natural language processing (NLP) tasks, including but not limited to:

- Named Entity Recognition (NER): Identifying and classifying named entities (e.g., persons, organizations, locations) in text.
- Token Classification: Assigning a label or category to each token (word or sub-word unit) in a given text input, useful for tasks like part-of-speech tagging, sentiment analysis, or named entity recognition.
- Question Answering (QA): Providing answers to questions based on the given context or knowledge base.
- Text Generation: Generating human-like text, such as stories, articles, or dialogue.
- Summarization: Producing a concise summary of a longer text while preserving the main ideas.
- Translation: Translating text from one language to another.

- Text Classification: Assigning a category or label to a given text input, useful for tasks like sentiment analysis, topic classification, or spam detection.
- Zero-Shot Classification: The ability to classify inputs into categories that the model has not been explicitly trained on, relying on its understanding of the task and the provided examples or descriptions.
- Feature Extraction: Extracting relevant features or representations from text data, which can be useful for downstream tasks like classification, clustering, or information retrieval.
Fill-Mask: Given a text input with one or more masked (or blanked-out) tokens, the model must predict the missing tokens based on the surrounding context.
Sentence Similarity: Determining the semantic similarity between two sentences or text inputs, useful for tasks like paraphrase detection, plagiarism detection, and text clustering

6. Token Classification:
Token classification is the task of assigning a label or category to each token (word or sub-word unit) in a given text input. This is useful for various NLP tasks, such as:

- Part-of-Speech (POS) Tagging: Assigning grammatical categories (noun, verb, adjective, etc.) to each token in a sentence.
- Named Entity Recognition (NER): Identifying and classifying named entities (e.g., persons, organizations, locations) in text.
- Sentiment Analysis: Determining the sentiment (positive, negative, or neutral) associated with each token or phrase in a text.

Token classification is important because it provides a more granular understanding of the text, enabling downstream applications like information extraction, text summarization, and question answering.

7. GPU Types:
GPUs (Graphics Processing Units) are specialized hardware accelerators widely used for running machine learning models, including LLMs. Different types of GPUs are available, each with varying performance and computational capabilities:

- Consumer-grade GPUs: These are GPUs designed for gaming and general-purpose computing, such as NVIDIA's GeForce and AMD's Radeon series. While capable of running machine learning models, they may not be optimized for this purpose.
- Professional GPUs: GPUs designed specifically for professional applications like scientific computing, data analytics, and machine learning. Examples include NVIDIA's Quadro and AMD's Radeon Pro series.
- Data Center GPUs: These are high-performance GPUs designed for use in data centers and cloud computing environments. Examples include NVIDIA's Tesla and AMD's Instinct series, which are optimized for deep learning and parallel computing workloads.
- Ampere GPUs: Ampere is NVIDIA's microarchitecture for their latest generation of data center GPUs, designed for high-performance computing, machine learning, and AI workloads. Some key features of Ampere GPUs include
- Specialized AI Accelerators: These are custom-designed chips specifically optimized for running machine learning models, such as NVIDIA's Tensor Core GPUs, Google's Tensor Processing Units (TPUs), and other AI-specific accelerators from companies like Graphcore, Habana, and Cerebras.

8. Base Models and Instruction Models:
In the context of LLMs, there are two main types of models:

- Base Models: These are the foundational language models trained on vast amounts of text data using self-supervised learning techniques like masked language modeling or autoregressive language modeling. Examples include GPT-3, BERT, RoBERTa, and XLNet. Base models are pre-trained on a broad corpus of text and can be fine-tuned or adapted for specific downstream tasks.
- Instruction Models: These are language models that have been further trained to follow instructions or prompts with high fidelity. Instruction models are trained on a dataset of instructions and demonstrations, allowing them to better understand and execute various tasks specified in natural language. Examples include InstructGPT, PaLM, and Claude (the AI assistant you're talking to). Instruction models are typically built on top of base models and can often perform tasks in a more controlled and reliable manner.

- Autoregressive models: Models like GPT-3 that generate text sequentially, one token at a time,- based on the previous context.
- Encoder-decoder models: Models like BART and T5 that use an encoder to process the input and a decoder to generate the output, useful for tasks like translation and summarization.
- Retrieval-augmented models: Models that combine retrieval from a knowledge base with language generation, like RAG (Retrieval-Augmented Generation).
- Multimodal models: Models that can process and generate data in multiple modalities, such as text, images, and audio, like DALL-E, Stable Diffusion, and Imagen.
- Domain-specific models: LLMs trained on data from specific domains, like biomedical literature, legal documents, or financial reports, for specialized applications.

9. Model Quantization and Saving Formats:
LLMs can be quantized and saved in various formats to reduce model size and improve inference efficiency. Some common formats include:

- GGML (Groovy Geometric Modeling Language): An open-source format developed by Anthropic for storing and quantizing LLMs. It supports 8-bit, 4-bit, and even 2-bit quantization.
- GGUF (Groovy Geometric Unified Format): An extension of GGML that allows storing multiple models in a single file, along with additional metadata and configurations.
- GPTQ (GPT Quantization): An open-source format and library for quantizing and compressing LLMs like GPT-2 and GPT-3. It supports various quantization techniques, including 8-bit quantization and quantization-aware training.
- AWQ (Accurate Weights Quantization): A quantization approach developed by Hugging Face that allows quantizing LLMs to lower precisions (e.g., 4-bit or 8-bit) while maintaining high accuracy.
- HuggingFace Serialization: HuggingFace provides its own serialization format for saving and loading pre-trained models, including LLMs. This format supports various compression and quantization techniques.
- TensorFlow SavedModel: TensorFlow's native format for serializing and saving trained models, including LLMs. It supports various compression and quantization techniques.
- ONNX (Open Neural Network Exchange): An open ecosystem for interoperable AI models, including support for quantized models and various compression techniques.

These formats often employ techniques like quantization, pruning, and sparse model representations to reduce the model size and improve inference efficiency, particularly for deployment on resource-constrained devices or edge computing scenarios.

10 .Calculation to check memory requirements based on parameters:

The memory required to store a neural network model depends primarily on the number of parameters and the precision used to represent them. Here's a simple formula to estimate the memory requirement:

Memory requirement (in bytes) = (Number of parameters) × (Precision in bytes)

For example, if a model has 1 billion parameters (1,000,000,000), and you want to store it using 16-bit precision (2 bytes per parameter), the memory requirement would be:

Memory requirement = 1,000,000,000 × 2 bytes = 2,000,000,000 bytes ≈ 1.86 GB