Assignment 7 — Solutions

LLM Fundamentals

Part B — Transformers & LLMs · 22 questions with answers

Q1. What is a Large Language Model? How is it trained (pre-training → instruction tuning → alignment)? ★ · Asked: Google

Answer

A **Large Language Model (LLM)** is a very large (billions of parameters) Transformer, usually decoder-only, trained to predict the next token on massive text data. Through this it learns grammar, facts, reasoning patterns and coding ability, and can perform many tasks through prompting.

Training stages:

1. **Pre-training** – self-supervised next-token prediction on trillions of tokens (web, books, code). Produces a **base model** that completes text. Most expensive stage (thousands of GPUs, weeks/months).
2. **Instruction tuning / SFT** – supervised fine-tuning on (instruction, response) pairs and multi-turn chats, so the model follows instructions and adopts a chat format.
3. **Alignment / preference tuning** – RLHF (reward model + PPO), DPO or similar methods using human (or AI) preference data, to make responses more helpful, honest and harmless. Recent models add **reinforcement learning on verifiable tasks** (maths, code) to improve reasoning.

Q2. What is the difference between a base model and an instruct/chat model? ★

Answer

- **Base model** – only pre-trained; it **continues text**. Ask "What is the capital of France?" and it may continue with more questions instead of answering. Useful as a starting point for your own fine-tuning, or with few-shot prompting.
- **Instruct / chat model** – base model + SFT + alignment. It **follows instructions**, answers questions, uses a chat template (system/user/assistant roles), refuses unsafe requests. Use this for applications.

Example names: Llama-3-8B (base) vs Llama-3-8B-Instruct.

Q3. What is RLHF? Explain the reward model and PPO. What is DPO and why is it simpler? ★ · Asked: Google

Answer

RLHF (Reinforcement Learning from Human Feedback):

1. **SFT** – fine-tune the base model on good demonstrations.
2. **Reward model** – humans compare pairs of responses to the same prompt (A better than B). Train a model to output a scalar score that predicts human preference.
3. **PPO (Proximal Policy Optimization)** – the LLM (policy) generates responses, the reward model scores them, and PPO updates the LLM to maximize reward, with a **KL penalty** that keeps it close to the SFT model (prevents "reward hacking" and gibberish). PPO clips updates for stability.

DPO (Direct Preference Optimization): skips the separate reward model and RL loop. It uses the preference pairs directly with a simple classification-style loss that increases the probability of the chosen response and decreases the rejected one, relative to a frozen reference model.

Why DPO is simpler: only two models (policy + reference) instead of four (policy, reference, reward, value), no sampling during training, stable like ordinary fine-tuning, fewer hyperparameters, cheaper. Variants: IPO, KTO, ORPO, SimPO.

Q4. What are scaling laws? What did the Chinchilla paper show?

Answer

Scaling laws (Kaplan et al., OpenAI 2020) show that LLM loss decreases **predictably as a power law** as you increase model parameters (N), training data (D) and compute (C). This lets labs predict the performance of a big model from small experiments.

Chinchilla (DeepMind, 2022) showed that earlier models (like GPT-3, Gopher) were **under-trained** – too many parameters for too little data. For a fixed compute budget, parameters and training tokens should be scaled **equally**; the optimum is roughly **20 tokens per parameter**. Chinchilla (70B params, 1.4T tokens) beat Gopher (280B params) with the same compute.

Today, models are often trained far beyond Chinchilla-optimal (e.g. Llama 3 8B on 15T tokens) because a smaller model trained longer is **cheaper at inference**.

Q5. What is a context window? What problems appear with very long contexts ("lost in the middle", cost, latency)? ★

Answer

The **context window** is the maximum number of tokens (prompt + generated output) the model can process at once – e.g. 8k, 128k, 200k or 1M+ tokens.

Problems with very long contexts:

- **"Lost in the middle"** – models use information at the beginning and end of the context better than information buried in the middle.
- **Cost** – API pricing is per token; huge prompts are expensive.
- **Latency** – time-to-first-token grows with prompt length (attention is $O(n^2)$, plus KV-cache memory).
- **Distraction** – irrelevant content can reduce accuracy; effective context is often shorter than the advertised limit.

Mitigations: RAG (send only relevant chunks), put key instructions at the start/end, summarization, prompt caching, re-ranking.

Q6. Explain decoding strategies: greedy, beam search, top-k, top-p (nucleus) and temperature. ★ · Asked: TCS, EPAM, Google

Answer

- **Greedy decoding** – always pick the highest-probability token. Fast, deterministic, but repetitive and can miss better sequences.
- **Beam search** – keep the top-B partial sequences at each step and return the best overall. Good for translation/summarization; tends to be generic and repetitive for open-ended chat.
- **Temperature** – divides logits by T before softmax. $T < 1 \rightarrow$ sharper, more deterministic; $T > 1 \rightarrow$ flatter, more random/creative; $T \rightarrow 0 \approx$ greedy.
- **Top-k sampling** – sample only from the k most likely tokens.

- **Top-p (nucleus) sampling** – sample from the smallest set of tokens whose cumulative probability ≥ p (e.g. 0.9); adapts the candidate set size to the model's confidence.

Other controls: repetition / frequency / presence penalties, min-p, max_new_tokens, stop sequences.

Q7. What is quantization (INT8, 4-bit, GPTQ, AWQ, GGUF)? What do you trade off? ★

Answer

Quantization stores model weights (and sometimes activations/KV cache) in lower precision – FP32 (4 bytes) → FP16/BF16 (2) → **INT8** (1) → **4-bit** (0.5 byte) – to reduce memory and speed up inference.

- **INT8 (LLM.int8())** – 8-bit with outlier handling; near-lossless.
- **4-bit NF4 (bitsandbytes)** – used by QLoRA for training on consumer GPUs.
- **GPTQ** – post-training quantization that minimizes layer-wise error using second-order information; fast GPU inference.
- **AWQ** – activation-aware; protects the ~1% most important weights; good quality at 4-bit.
- **GGUF** – file format used by **llama.cpp / Ollama** for CPU and Apple-silicon inference, with many quant levels (Q4_K_M, Q5_K_M, Q8_0).

Trade-off: less memory, lower cost and faster inference vs some **loss in accuracy** (small at 8-bit, noticeable at 3-bit and below, especially on maths/reasoning), plus calibration effort and hardware/kernel support.

Q8. How much GPU memory is needed to load a 7B model in FP16? In 4-bit? (Show the calculation.)

Answer

Memory for weights = parameters × bytes per parameter.

- **FP16:** $7 \times 10^9 \times 2$ bytes = **14 GB**
- **8-bit:** $7 \times 10^9 \times 1$ byte = **7 GB**
- **4-bit:** $7 \times 10^9 \times 0.5$ byte = **3.5 GB**

Add roughly **10–20% overhead** for the KV cache, activations and CUDA context (more for long contexts/large batches). So a 7B model in FP16 needs a ~16 GB GPU; in 4-bit it fits in ~5–6 GB.

Full fine-tuning with Adam in mixed precision needs about **16 bytes per parameter** (weights + gradients + 2 optimizer states + FP32 master copy) ≈ 112 GB for 7B – which is why LoRA/QLoRA are used.

Q9. What is knowledge distillation? How are small models trained from large ones?

Answer

Knowledge distillation trains a small **student** model to imitate a large **teacher** model.

- **Logit/soft-label distillation:** the student is trained to match the teacher's full probability distribution (soft targets, often with temperature) using KL-divergence loss, plus the normal loss on true labels. Soft targets carry richer information ("this is mostly a cat, a little like a dog"). Example: **DistilBERT** – 40% smaller, 60% faster, keeps ~97% of BERT's performance.
- **Data/sequence distillation (common for LLMs):** use a large model to **generate high-quality training data** (answers, reasoning traces), then fine-tune the small model on it. Examples: Alpaca, Orca, distilled reasoning models (DeepSeek-R1-Distill).

Benefits: cheaper and faster deployment, on-device models. (Check the teacher's licence terms before using its outputs for training.)

Q10. What are emergent abilities and in-context learning?

Answer

- **Emergent abilities** – abilities that are absent or near-random in smaller models but appear (seemingly suddenly) once models pass a certain scale: multi-step arithmetic, chain-of-thought reasoning, following complex instructions, code generation. (Some researchers argue the "suddenness" is partly an artefact of the evaluation metrics chosen.)
- **In-context learning (ICL)** – the ability to **learn a task from examples or instructions given in the prompt, without updating any weights**. Example: give 3 examples of "review → sentiment" and the model labels a 4th correctly. First highlighted with GPT-3 ("Language Models are Few-Shot Learners").

Q11. What are reasoning models (models that "think" before answering)? When are they worth the extra latency and cost?

Answer

Reasoning models (e.g. OpenAI o-series, DeepSeek-R1, Claude/Gemini with extended "thinking") are trained, often with reinforcement learning on verifiable problems, to **generate a long internal chain of thought before giving the final answer** – exploring approaches, checking work and correcting mistakes. They spend more compute at inference time ("test-time compute scaling").

Worth it for: complex maths, multi-step logic, hard coding/debugging, planning, agentic tasks, analysis where accuracy matters more than speed.

Not worth it for: simple lookups, classification, extraction, chit-chat, high-volume low-latency tasks – there they add latency (seconds to minutes) and cost (many extra tokens) without improving results. A common pattern is to **route** easy queries to a fast model and hard ones to a reasoning model.

Q12. What is hallucination in an LLM? What causes it and how do you reduce it? ★

· Asked: TCS, Infosys, EPAM, Google

Answer

Hallucination – the model generates content that is fluent and confident but **factually wrong, unsupported or made up** (fake citations, wrong numbers, non-existent APIs).

Causes: LLMs are trained to produce *plausible* next tokens, not verified facts; gaps or errors in training data; knowledge cut-off; ambiguous prompts; sampling randomness; training that rewards answering over saying "I don't know".

How to reduce:

- **RAG** – ground answers in retrieved documents and instruct "answer only from the context; say 'I don't know' otherwise".
- Ask for **citations/quotes** from sources and verify them.
- Lower temperature for factual tasks.
- Clear, specific prompts; chain-of-thought or self-verification steps.
- **Tools** (search, calculator, database) for facts and maths.
- Fine-tuning on domain data; better/larger models.
- Evaluation and guardrails: faithfulness checks (RAGAS), LLM-as-judge, human review for high-risk outputs.

Q13. Open-source vs closed (API) LLMs — how do you choose for an enterprise use case (cost, privacy, quality, control)?

Answer

Closed/API models (GPT-4o, Claude, Gemini): best quality and reasoning, no infrastructure, fast to start, frequent upgrades. Concerns: data leaves your environment (check enterprise terms, data residency), per-token cost at scale, vendor lock-in, rate limits, model changes outside your control.

Open-weight models (Llama, Mistral, Qwen, Gemma): full control and privacy (on-prem/VPC), can fine-tune deeply, predictable cost at high volume, no lock-in. Concerns: you manage GPUs, scaling, security and updates; usually somewhat lower quality on the hardest tasks; licence restrictions.

How to choose for an enterprise (e.g. a bank): data sensitivity/regulation, required quality (run an eval on your own task), volume and latency, budget, in-house MLOps skills, need for fine-tuning. Common answer: **hybrid** – API models for complex, non-sensitive tasks; self-hosted open models for sensitive data or high-volume simple tasks; abstract the provider behind one interface (e.g. LangChain/LiteLLM) so you can switch.

Q14. What is the difference between an LLM (text-completion) interface and a Chat model interface?

Answer

- **LLM (text-completion) interface** – input is a **single string**, output is a string that **continues** it. Example: prompt "The capital of France is" → " Paris". In LangChain: `LLM` classes, `.invoke("text")` returns a string.
- **Chat model interface** – input is a **list of messages with roles** (system, user/human, assistant/AI, tool), output is an **AIMessage**. Supports multi-turn conversation, system instructions, tool calling and structured outputs. In LangChain: `ChatOpenAI`, `ChatAnthropic`, etc.

Almost all modern models are used through the chat interface; the chat template is converted to a single token sequence internally.

Q15. What is multimodality? Name a few multimodal models and explain how images are fed to them. ★ · Asked: Google

Answer

Multimodality – a model that can take in and/or generate **more than one type of data**: text, images, audio, video.

Examples: GPT-4o, Gemini (1.5/2.x), Claude 3+/4 family (vision), Llama 3.2 Vision, Qwen-VL, LLaVA, Pixtral; Whisper (audio → text); DALL·E, Stable Diffusion (text → image).

How images are fed: a **vision encoder** (ViT/CLIP-style) splits the image into patches (e.g. 14×14 pixels) and turns them into embeddings; a **projection layer/adapter** maps these into the LLM's token embedding space; the resulting "image tokens" are placed in the sequence alongside text tokens and the LLM attends to both.

In code, you send the image as **base64 or a URL** inside the message content:

```
from langchain_core.messages import HumanMessage
msg = HumanMessage(content=[
    {"type": "text", "text": "Describe this chart"},
    {"type": "image_url", "image_url": {"url": f"{b64}"}}])
llm.invoke([msg])
```

Q16. How are image generation (diffusion) models different from LLMs?

Answer

- **LLMs** are **autoregressive**: they generate discrete **tokens one at a time**, each conditioned on previous tokens, using a Transformer trained with next-token cross-entropy.
- **Diffusion models** (Stable Diffusion, DALL·E 3, Imagen) generate **continuous data (images)** by **iterative denoising**: in training, noise is gradually added to images and a network (U-Net or Diffusion Transformer) learns to predict/remove the noise; at generation, they start from pure random noise and denoise it over many steps (20–50) into an image. Text conditions the process through **cross-attention** with a text encoder (CLIP/T5). Latent diffusion works in a compressed latent space (via a VAE) for efficiency.

Key differences: discrete tokens vs continuous pixels/latents; sequential token generation vs refining the whole image at once over many steps.

Q17. What is a GAN? Explain the roles of the generator and discriminator.

Answer

A **GAN (Generative Adversarial Network)**, Goodfellow 2014, has two networks trained against each other:

- **Generator (G)** – takes random noise z and produces fake data (e.g. an image). Goal: **fool the discriminator**.
- **Discriminator (D)** – a binary classifier that receives real and generated samples and predicts real vs fake. Goal: **catch the fakes**.

Training (min-max game): alternate – update D to better separate real and fake; update G (through D's gradients) so D labels its fakes as real. At equilibrium, G produces realistic samples and D outputs ~0.5.

Uses: image synthesis (StyleGAN), super-resolution, image-to-image translation (CycleGAN, Pix2Pix), synthetic data (e.g. for fraud datasets). Problems: **mode collapse**, unstable training, hard to evaluate. Diffusion models have largely replaced GANs for image generation.

Q18. Generative vs discriminative models — explain with examples.

Answer

- **Discriminative models** learn the **decision boundary / P(y | x)** – "given this input, which class?". Examples: logistic regression, SVM, random forest, BERT classifier. Usually better for pure classification.
- **Generative models** learn the **data distribution P(x) or P(x, y)** – "how is the data produced?" – so they can **generate new samples**. Examples: Naive Bayes, Hidden Markov Models, GANs, VAEs, diffusion models, GPT-style LLMs.

Example: for spam detection, a discriminative model learns what separates spam from non-spam; a generative model learns what spam emails look like and could write a new one.

Q19. What is a token? Roughly how many tokens is a word or a page, and why does token count matter? ★ · Asked: Google

Answer

A **token** is the unit an LLM reads and writes – a word, part of a word, punctuation or space, produced by the model's tokenizer (BPE etc.).

Rule of thumb (English): 1 token ≈ 4 characters ≈ 0.75 words → 100 tokens ≈ 75 words; one A4 page of text ≈ 500–700 tokens. Hindi/Marathi and other non-Latin scripts often need **more tokens per word**.

Why it matters:

- **Cost** – APIs charge per input and output token.
- **Latency** – generation time grows with output tokens; prompt processing with input tokens.
- **Context window limits** – prompt + history + retrieved chunks + answer must fit.
- Chunk sizes for RAG are usually defined in tokens.

Count with the model's tokenizer (e.g. `tiktoken` for OpenAI).

Q20. Can hallucinations be completely eliminated from a GenAI application? ★ ·

Asked: Deloitte, Google

Answer

No. LLMs generate text by predicting likely tokens, so there is always some probability of a fluent but wrong statement. They can be **reduced to an acceptable level and managed**, not eliminated:

- Ground answers with **RAG** and instruct "answer only from context, otherwise say I don't know".
- Low temperature, clear prompts, structured outputs.
- **Citations** so users can verify; tools for facts and calculations.
- **Validation layers:** faithfulness checks, rules/regex for numbers and IDs, LLM-as-judge.
- **Human-in-the-loop** for high-risk outputs; confidence thresholds and fallbacks.
- Continuous evaluation and monitoring of hallucination rate.

The right answer in interviews: be honest about the limitation and explain the **risk-based controls** you put around it.

Q21. What is a foundation model, and what does "alignment" mean for LLMs? ·

Asked: Google

Answer

- **Foundation model:** a large model pre-trained on broad data at scale (text, code, images) that can be **adapted to many downstream tasks** via prompting, RAG or fine-tuning – e.g. GPT-4o, Gemini, Claude, Llama. One base model, many applications.
- **Alignment:** making the model's behaviour match human intentions and values – **helpful** (follows instructions), **honest** (doesn't make things up, admits uncertainty) and **harmless** (refuses dangerous requests). Achieved with instruction tuning (SFT), RLHF/DPO, constitutional/AI-feedback methods, safety training and red-teaming.

Q22. How would you choose an LLM for a production application, and how do you compare two LLMs? ★ · Asked: Google

Answer

Selection criteria:

- **Quality on YOUR task** – build an evaluation set and test candidates; public benchmarks are only a first filter.
- **Latency** (time-to-first-token, tokens/sec) and **cost** per request at expected volume.
- **Context window**, multimodal needs, tool calling and structured-output support, languages.
- **Data privacy/compliance** – data residency, retention, on-prem need → open-weight vs API.
- Reliability/rate limits, vendor support, licence, fine-tuning options.

Comparing two LLMs: run both on the same golden dataset with the same prompts; score with task metrics + LLM-as-judge (pairwise, order-swapped) + human review of a sample; compare latency and cost; check failure cases (hallucination, format errors, refusals); then A/B or shadow

test in production. Often the answer is a **router**: a cheap model for easy queries and a strong model for hard ones.