Contents

Chapter 2 (Extended): Foundation Models — The Missing Topics

> **Companion to** `Chapter_02_Foundation_Models.md`. This supplement fills the gaps between the main notes and the actual lecture slides (TU Braunschweig, AI Engineering, WS 25/26). It does **not** repeat what the main file already covers well — it adds the topics that appear on the slides but were thin or absent: data bias, vocabulary granularity & API cost, the context window, one-hot vs. learned embeddings & Word2Vec, why attention is an *advantage*, Pre- vs. Post-LayerNorm, the entire training-systems story (FLOPs, CPU vs. GPU, precision formats, the H100, distributed training), SFT limitations & the two-stage paradigm, the Kaplan-vs-Chinchilla story with *compute-optimal ≠ deploy-optimal*, and the full evaluation taxonomy (intrinsic, BLEU, ROUGE, Pass@k, MMLU, BIG-Bench, HumanEval, practical-capability, safety, τ^2-bench).
>
> Each section follows the same rhythm as the main notes: **what the slide says → intuition → worked detail → বাংলা ব্যাখ্যা → exam-style questions → MCQs (answer + why).**
>
> PDFs mapped: `Ch2.1-2.3.pdf` (slides 37–52), `Ch2.4-2.5part1.pdf` (slides 56–64), `Ch2.5part2.pdf` (slides 89–100), `Ch2.5part3-Ch2.7.pdf` (slides 124–127), `Ch2.8-Ch2.10part1.pdf` (slides 150–175), `Ch2.10part2-Ch2.12part1.pdf` (slides 191–209), `Ch2.12part2-Ch4part1.pdf` (slides 210–218).

বাংলা ব্যাখ্যা: এই ফাইলটা মূল Chapter 2 নোটের **পরিপূরক** — যেসব টপিক স্লাইডে আছে কিন্তু মূল নোটে কম/নেই, শুধু সেগুলোই এখানে গভীরভাবে আছে প্রতিটা টপিকে: স্লাইড কী বলছে → স্বজ্ঞা (intuition) → হাতে-কলমে ডিটেইল → বাংলা সারাংশ → পরীক্ষার প্রশ্ন → MCQ (উত্তর + কেন)

How the exam tests this material (quick reminder)

Question type	What to write
Multiple choice	exactly **one** correct option — "Which statement best describes…"
"Explain why…" (3 pt)	**cause → mechanism → consequence**, full technical terms, no abbreviations
Application (5 pt)	name the **technique** → **justify** → state a **trade-off**
Numeric	round to **2 decimal places**, non-programmable calculator

বাংলা ব্যাখ্যা: নিচের প্রতিটা টপিক এই চারটা ফরম্যাটেই পরীক্ষায় আসতে পারে তাই ব্যাখ্যাগুলো "কারণ → কীভাবে → ফলাফল" কাঠামোতে সাজানো

2.1+ — Training Data: Bias (the missing half of Section 2.1)

What the lecture says

The model's only goal is to **minimize next-token prediction error**. It therefore *replicates* the statistical patterns of its data — it never evaluates whether those patterns are true or fair. Three ideas stack on top of each other:

1. **LLMs learn correlations, not truth.** Any asymmetry in the data distribution becomes part of the learned representation.
2. **Bias is a property of the data distribution.** Internet-scale corpora mix news, forums, and books, but *not equally* — some voices dominate, so predictions favour overrepresented groups and perspectives.
3. **Training amplifies bias.** A *small* statistical asymmetry (e.g. men mentioned slightly more often in leadership contexts) becomes a *stronger, more systematic* pattern in outputs. The model does not merely reflect what it saw — it **generalizes and reinforces** the correlation.

The canonical example — language bias in GPT-4

The same model (GPT-4) was evaluated on **MMLU** translated into many languages. Result: it scores **highest in English**. Why? English is overrepresented in the training data, and some languages are also intrinsically harder to model (richer morphology, less standardized text). The capability gap is therefore a *data* artefact, not a *reasoning* artefact — the underlying model is identical.

Intuition

A model is a mirror that has been polished unevenly: it reflects the crowd that talked the most. If 80 % of leadership text said "he", the model's prior for a CEO pronoun is not 80 % — training pushes it *past* the data frequency, because the optimizer rewards confident, low-loss predictions.

বাংলা ব্যাখ্যা: মডেল সত্য শেখে না, **সম্পর্ক (correlation)** শেখে ডেটায় যে দল বেশি কথা বলেছে, মডেল সেই দলের দিকে ঝুঁকে পড়ে — আর ট্রেনিং সেই সামান্য ঝোঁককে **আরও বাড়িয়ে** দেয় (amplification) GPT-4 ইংরেজিতে সবচেয়ে ভালো, কারণ ট্রেনিং-ডেটায় ইংরেজি বেশি — এটা মডেলের বুদ্ধির সীমা নয়, ডেটার ভারসাম্যহীনতার ফল

Exam-style questions

- **Explain why** an LLM can *amplify* a bias that is only mild in its training data. *(cause: optimizer rewards confident low-loss predictions → mechanism: it generalizes the majority correlation and pushes probability past the raw data frequency → consequence: a mild asymmetry becomes a systematic output pattern.)*
- **Application:** A bank wants to use an LLM to screen loan applications. Name one risk rooted in training-data bias, justify it, and state a mitigation trade-off. *(risk: disparate impact on underrepresented groups; justify: data reflects historical lending bias; trade-off: filtering/Reweighting reduces bias but can lower accuracy on the majority distribution.)*

MCQs

Q1. Which statement best describes why LLMs reproduce social bias? A. They are explicitly programmed with biased rules. B. Their training objective rewards replicating data patterns, not judging their fairness. ✓ C. Bias is added during tokenization. D. Larger models are immune to bias. *Why:* the next-token objective optimizes pattern replication; fairness is never in the loss. A/C/D misattribute the source.

Q2. GPT-4 scores highest on English MMLU primarily because: A. English grammar is objectively simpler. B. English is overrepresented in the training data. ✓ C. The benchmark was written by English speakers. D. The model uses a different architecture for English. *Why:* the model is identical across languages; the differentiator is data quantity (with language difficulty a secondary factor).

Q3. "Amplification" of bias means that: A. The model stores bias in a dedicated parameter. B. A small data asymmetry becomes a stronger, more systematic output pattern. ✓ C. Bias only appears after fine-tuning. D.

The dataset is made larger. *Why:* amplification is about the model generalizing and reinforcing a mild correlation, not about storage or dataset size.

2.2+ — Tokenization: Vocabulary Granularity, Impact, and API Cost

Three granularity levels (memorize the trade-offs)

Level	Example for "Hello"	Pros	Cons
Character	`H e l l o`	tiny vocabulary; works for any language/typo; **no unknown words**	very long sequences; hard to capture word meaning
Subword (BPE)	`Hel lo`	balance of flexibility & efficiency; handles rare words by composition	slightly harder to train; depends on corpus statistics
Word	`Hello`	intuitive, short sequences; direct mapping to meaning	huge vocabulary; **cannot handle unseen words**; memory heavy

Modern LLMs use **subword** tokenization (byte-level BPE) — it is the only level that is simultaneously compact, OOV-free, and short-sequence.

Why vocabulary size matters (three effects)

1. **Sequence length & compute.** Smaller vocabulary → more tokens per sentence → longer sequences → higher cost (attention is quadratic in sequence length).
2. **Learning quality.** Each token is a unit of meaning the model must learn. Frequent tokens get better representations; rare ones stay noisy. Bad segmentation (`compu` + `ter`) blurs meaning vs. a clean `computer`.
3. **Memory.** Each token type needs its own embedding row, so a larger vocabulary directly increases model size.

The design question the slides pose: *how do we automatically build a vocabulary that keeps sequences short, covers any word, and stays compact?* → the answer is **(byte-level) BPE**.

API cost — the practical consequence

Different providers use **different tokenizers**, so the *same* sentence becomes a *different* number of tokens, and the **effective cost per word varies** between providers even at the same price-per-token. Two takeaways the slide stresses:

- The "next-word prediction" problem is really **next-token prediction**; vocabulary size = the model's **output dimension** (it predicts a probability over every vocabulary entry).
- **Tokenizer and trained model are inseparable** — you cannot swap the tokenizer after training, because every embedding row and the output layer are tied to specific token IDs.

Worked example — token count drives cost

Sentence: "internationalization is hard". Suppose tokenizer A (large vocab) emits 3 tokens; tokenizer B (small vocab) emits 7 tokens. At \$10 per million tokens, processing 1 million such sentences costs A = 3 M × \$10/M = **\$30** and B = 7 M × \$10/M = **\$70** — the *same text*, 2.33× the cost, purely from tokenization granularity.

বাংলা ব্যাখ্যা: ছোট ভোকাবুলারি → প্রতি বাক্যে বেশি টোকেন → লম্বা সিকোয়েন্স → বেশি খরচ ও ধীর গণনা বড় ভোকাবুলারি → কম টোকেন কিন্তু বেশি মেমরি (প্রতিটি টোকেনের আলাদা embedding row) তাই subword (BPE) হলো সোনার মাঝামাঝি পথ মনে রেখো: **টোকেনাইজার আর মডেল একসাথে বাঁধা** — ট্রেনিং-এর পরে টোকেনাইজার বদলানো যায় না

Exam-style questions

- **Explain why** two API providers can charge different effective prices for the identical input text. *(cause: each uses a different tokenizer → mechanism: the same text maps to a different token count → consequence: tokens billed differ, so effective $/word differs.)*
- **Application:** You must process huge volumes of German legal text cheaply. Which tokenizer property do you optimize, and what is the trade-off? *(optimize: high tokens-per-word efficiency for German → fewer tokens, lower cost; trade-off: a German-tuned vocabulary may be larger / less efficient for other languages and increases embedding memory.)*

MCQs

Q4. Compared with word-level tokenization, character-level tokenization mainly: A. increases vocabulary size. B. eliminates unknown words but lengthens sequences. ✓ C. reduces compute cost. D. makes the model bigger. *Why:* characters cover everything (no OOV) but produce long sequences; vocabulary shrinks, not grows.

Q5. Why can a tokenizer **not** be swapped after the model is trained? A. The license forbids it. B. Embedding rows and the output layer are bound to specific token IDs. ✓ C. It would change the learning rate. D. Tokenizers are encrypted. *Why:* every parameter that touches token identity (input embeddings, output projection) is keyed to the trained vocabulary.

Q6. A smaller vocabulary tends to **increase** which cost? A. Embedding-matrix memory. B. Per-sequence compute, because sequences get longer. ✓ C. The irreducible loss. D. Disk size of the tokenizer. *Why:* fewer token types → more tokens per sentence → longer sequences → more (quadratic) attention compute.

Q7. "Vocabulary size = output dimension" means: A. the model outputs one number total. B. the final layer produces a probability distribution over all vocabulary entries. ✓ C. the input is one-hot only. D. the context window equals the vocabulary. *Why:* the model predicts P(next token) over the whole vocabulary, so the output layer has one logit per token type.

2.3+ — The Context Window

What the lecture says

The **context window length** is the maximum number of tokens an LLM can process at once. Key properties:

- It is the model's **short-term memory** and bounds **input + generated** tokens together.
- During inference, each new token is predicted **only** from previous tokens *within the window.*
- It is a **fixed architectural property** — set at design/training time, not a runtime knob.
- Larger windows increase **latency, memory use, and cost**.

When the limit is hit, older tokens are handled by one of: **dropping** them, **replacing them with a summary**, or **moving them into a database (RAG)** — foreshadowing Chapter 4.

Model	Context window (tokens)
GPT-1	512
GPT-2	1024
GPT-3	2048
GPT-4	32,768

Intuition

The context window is the size of the desk you can spread papers on. Anything you push off the edge is gone unless you first summarize it onto a sticky note (summary) or file it in a cabinet you can fetch from later (RAG).

বাংলা ব্যাখ্যা: কনটেক্সট উইন্ডো = মডেলের **স্বল্পমেয়াদি স্মৃতি**, অর্থাৎ একসাথে কত টোকেন (ইনপুট + আউটপুট) দেখতে পারে তার সর্বোচ্চ সীমা এটা **স্থাপত্যগত (architectural)** ব্যাপার, রানটাইমে বদলানো যায় না সীমা পেরোলে পুরোনো টোকেন: বাদ দেওয়া হয় / সারাংশে রূপান্তর হয় / ডেটাবেসে (RAG) পাঠানো হয় বড় উইন্ডো = বেশি খরচ ও latency

Exam-style questions

- **Explain why** doubling the context window more than doubles the compute per step. *(cause: self-attention compares every token with every other → mechanism: cost grows with the square of sequence length → consequence: 2× tokens ≈ 4× attention compute.)*
- **Application:** A chatbot must "remember" a 200-page manual that exceeds its context window. Name the technique, justify it, state a trade-off. *(technique: retrieval-augmented generation; justify: store the manual externally and fetch only relevant chunks per query; trade-off: retrieval errors / extra latency vs. fitting everything in-context.)*

MCQs

Q8. The context window is best described as: A. the model's long-term memory stored in weights. B. the maximum number of tokens (input + output) processable at once. ✓ C. the vocabulary size. D. the number of layers. *Why:* it bounds the combined input+generated token count for a single forward/generation pass.

Q9. When the context window overflows, which is **not** a standard strategy from the slides? A. Drop the oldest tokens. B. Summarize older tokens. C. Move older tokens into a database (RAG). D. Automatically expand the window at runtime. ✓ *Why:* the window is a fixed architectural property; you cannot simply enlarge it at inference — you manage the overflow instead.

Q10. Larger context windows primarily increase: A. irreducible loss. B. latency, memory, and cost. ✓ C. vocabulary size. D. the learning rate. *Why:* more tokens in attention means more memory and compute, hence higher latency and cost.

2.4+ — One-Hot vs. Learned Embeddings, and Word2Vec

One-hot representation and its limits

The simplest numeric encoding gives each token a unique position in a vocabulary-length vector. For vocabulary `["The","bank","raised","interest","rates","today"]`, "bank" = `[0,1,0,0,0,0]`. Three fatal limits:

1. **Dimensionality explosion** — each vector is mostly zeros; storage and compute are wasteful at vocabulary sizes of 100k+.
2. **All vectors are orthogonal** — every token is equidistant from every other, so the encoding carries **no relation** between words (`bank` is as close to `loan` as to `banana`).
3. **Poor generalization** — learning must start from scratch for every token; nothing is shared.

The embedding layer is a linear projection of the one-hot vector: it reduces dimensionality from `V` (vocab) to `d` (embedding size) and introduces **shared parameters**. Conceptually, one-hot × embedding-matrix = "pick that token's row".

The embedding matrix self-organizes meaning

The matrix `E` $\in$ $\mathbb{R}$`^{V×d}` is trained with all other parameters by backpropagation. Crucially: **the model does not store meaning in words — it stores meaning in the *distances and directions* between vectors.** Over billions of examples, tokens appearing in similar contexts get similar vectors, and an *emergent geometry* appears: gender (king–queen, man–woman), tense (walk–walked), syntax (is–was, do–did).

A subtle, exam-worthy fact: norm ≈ frequency, direction ≈ meaning

In embedding space, **word frequency correlates with vector norm**:

- **Frequent words** ("the", "and", "of") get **larger norms** (more gradient updates) but point in **similar directions**, forming a dense cluster near the centroid — shared generic usage, not meaning.
- **Rare words** ("cryptocurrency", "neutrino") get **smaller norms** but occupy **distinct directions** in sparser, peripheral regions.

Mnemonic: **vector *length* ≈ generality/frequency; vector *direction* ≈ meaning.** This is exactly why **cosine similarity** (direction only, length ignored) is the right tool — "car" and "automobile" share meaning regardless of how often each appears.

Word2Vec — the historical precursor

- **Objective:** predict context words from a center word (**skip-gram**).
- **Architecture:** a *shallow* network — one-hot input → hidden layer → output probability distribution over the vocabulary.
- **Training data:** word pairs within a context window. For "The bank raised interest rates today", center `interest` gives pairs `(interest→bank)`, `(interest→raised)`, `(interest→rates)`, `(interest→today)`; training **maximizes the probability** of the true context words.

Word2Vec embeddings are *static* (one vector per word, no context). Transformer embeddings are the *input layer* of a model that then makes them **context-dependent** via attention — that is the leap from 2.4 to 2.5.

বাংলা ব্যাখ্যা: One-hot ভেক্টরে প্রতিটি শব্দ আলাদা অক্ষে — সব শব্দ একে অপরের থেকে **সমান দূরত্বে**, তাই কোনো সম্পর্ক বোঝা যায় না, আর মাত্রা বিশাল Embedding হলো এই one-hot-এর **linear projection** — ছোট মাত্রায় নামিয়ে শেয়ার্ড প্যারামিটার আনে গুরুত্বপূর্ণ: মডেল অর্থ রাখে শব্দে নয়, ভেক্টরের **দূরত্ব ও দিকের** মধ্যে মনে রাখার সূত্র: **দৈর্ঘ্য ≈ কত সাধারণ/ঘনঘন, দিক ≈ অর্থ** — তাই cosine (শুধু দিক) দিয়ে অর্থের মিল মাপা হয় Word2Vec (skip-gram) এই ধারণার পূর্বপুরুষ, কিন্তু তার ভেক্টর **স্থির** (context ছাড়া)

Worked example — why cosine, not Euclidean, for "car" vs "automobile"

Let `car = [4, 0]` (rare, small-ish), `automobile = [8, 0]` (same direction, larger norm because more frequent in this toy). Euclidean distance = `|8−4| = 4.00` (looks "far"). Cosine similarity = `(4·8 + 0)/(4 · 8) = 32/32 = 1.00` (identical meaning). Direction captures the synonymy; length would have wrongly separated them.

Exam-style questions

- **Explain why** one-hot vectors cannot express that "bank" and "loan" are related. *(cause: each token sits on its own axis → mechanism: all one-hot vectors are mutually orthogonal → consequence: every pair has identical (zero) similarity, so no relation can be encoded.)*
- **Application:** You observe that the very frequent token "the" has a huge embedding norm yet is semantically empty. Which similarity measure protects your retrieval system, and what is the trade-off? *(measure: cosine similarity, which ignores magnitude; trade-off: you discard frequency information that might occasionally be useful, e.g. for weighting.)*

MCQs

Q11. The biggest representational limitation of one-hot vectors is that: A. they are too small. B. they are all orthogonal, encoding no relationship between tokens. ✓ C. they require GPUs. D. they change during inference. *Why:* orthogonality means equal distance between every pair — no semantic structure.

Q12. An embedding layer can be viewed as: A. a non-linear activation. B. a linear projection that maps a one-hot vector to its dense row. ✓ C. a tokenizer. D. a normalization step. *Why:* one-hot × E selects (projects to) the token's learned vector; it reduces V→d with shared parameters.

Q13. In trained embedding space, a token's vector **norm** most closely reflects its: A. meaning. B. position in the sentence. C. training frequency / generality. ✓ D. part of speech. *Why:* frequent words accumulate more gradient updates → larger norms; *direction* carries meaning.

Q14. Word2Vec's skip-gram objective is to: A. predict the center word from its context. B. predict context words from the center word. ✓ C. translate between languages. D. compress the vocabulary. *Why:* skip-gram maximizes P(context | center); (predicting center from context is the CBOW variant).

Q15. Why is cosine similarity preferred over Euclidean distance for comparing word meanings? A. It is faster to compute. B. It compares direction and ignores magnitude/frequency. ✓ C. It always returns positive values. D. It needs no normalization. *Why:* meaning lives in direction; cosine ignores the norm so synonyms of different frequency still match.

2.5+ — Why Attention Is an *Advantage*, and "Semantic Distortion"

The main notes explain the *mechanics* of attention (Q/K/V, scaled dot-product, multi-head). The slides also argue *why* attention beats the old RNN seq2seq — this is prime "Explain why" material.

The four advantages of the attention mechanism

1. **Solves the information bottleneck.** Classic RNN seq2seq squeezes the whole source sentence into one fixed-length context vector. Attention replaces it with a **dynamic, content-dependent** representation: each output token attends to a *different subset* of encoder states, bypassing the compression limit.
2. **Mitigates vanishing gradients.** Attention creates direct **shortcut connections** between distant tokens, so gradients flow easily from output back to early inputs — deeper sequence models train more stably.
3. **More "human-like" processing.** It mimics how humans re-examine the input selectively instead of relying on a single memory of the whole sentence.
4. **Interpretability / soft alignment.** Attention weights reveal which input tokens influenced each output token — visualizable as **attention maps**, giving a learned, unsupervised soft alignment between input and output words.

"Semantic Distortion?" — does positional encoding ruin meaning?

A natural worry: adding a positional vector changes each embedding's **direction (angle)**, so does it corrupt the word's meaning? The slide's answer is **no**:

- **Embedding magnitudes** $\gg$ **positional magnitudes**, so the direction changes only slightly — the vector stays within its semantic neighbourhood.
- The vector now encodes **"what the token is" + "where it occurs"**. The Transformer does not need pure word meaning; it needs the **contextualized meaning at a specific position**.
- The model **learns during training to separate the "what" from the "where"** components.

Result: every token becomes "a word at a place", which is exactly what lets attention reason about **order**.

বাংলা ব্যাখ্যা: RNN seq2seq পুরো বাক্যকে একটা স্থির ভেক্টরে চাপে → তথ্য হারায় (bottleneck) Attention প্রতিটি আউটপুট টোকেনকে ইনপুটের আলাদা আলাদা অংশে তাকাতে দেয় → bottleneck নেই, distant টোকেনে সরাসরি gradient যায় (vanishing gradient কমে), আর attention map দেখে বোঝা যায় কে কার দিকে তাকিয়েছে (interpretability) Positional encoding অর্থ নষ্ট করে না, কারণ embedding-এর দৈর্ঘ্য positional ভেক্টরের চেয়ে অনেক বড় — দিক সামান্যই বদলায়, আর মডেল "কী" বনাম "কোথায়" আলাদা করতে শেখে

Exam-style questions

- **Explain why** RNN-based seq2seq struggles with long sentences while attention does not. *(cause: a single fixed-length context vector must hold the entire source → mechanism: long inputs overflow this fixed capacity (the information bottleneck) → consequence: detail is lost; attention removes the bottleneck with a dynamic per-token representation.)*
- **Application:** A translation model loses the subject of long German sentences. Name the mechanism that fixes this, justify it, state a trade-off. *(mechanism: attention / self-attention; justify: lets each output token attend directly to the relevant source token regardless of distance; trade-off: $O(n^2)$ compute in sequence length.)*

MCQs

Q16. Attention's main fix for RNN seq2seq is that it: A. removes positional information. B. replaces a fixed-length context vector with a dynamic, content-dependent one. ✓ C. shrinks the vocabulary. D. eliminates the need for embeddings. *Why:* the dynamic representation defeats the information bottleneck of a single context vector.

Q17. Attention helps with vanishing gradients because it: A. normalizes activations. B. provides direct shortcut connections between distant tokens. ✓ C. uses a smaller learning rate. D. reduces the number of layers. *Why:* short paths between far-apart tokens let gradients flow back without decaying through many recurrent steps.

Q18. Why does adding positional encoding **not** destroy a token's meaning? A. Positional vectors are zero. B. Embedding magnitude is much larger than positional magnitude, so direction shifts only slightly. ✓ C. Meaning is stored in the tokenizer. D. Positional encoding is removed before attention. *Why:* the small positional perturbation keeps the vector in its semantic neighbourhood while adding "where" information.

Q19. Attention maps are valuable mainly for: A. reducing memory. B. interpretability — showing which inputs influenced each output (soft alignment). ✓ C. speeding up decoding. D. compressing the model. *Why:* the weights expose a learned, unsupervised alignment between input and output tokens.

2.6+ — Pre-LayerNorm vs. Post-LayerNorm

A high-yield architectural detail: *where* you place the normalization changes whether a deep Transformer trains at all.

Post-LayerNorm (the original "Attention Is All You Need")

Layout: `LayerNorm(x + Sublayer(x))` — the residual path **passes through** LayerNorm.

- Because normalization sits on the residual path, it **alters the gradient signal**: across many blocks, gradients get weaker, stronger, rotated, or mixed across dimensions.
- The **identity mapping becomes impossible** — the model can never simply "copy" its input through a block.
- Consequence: **deep Transformers become unstable**; hard to train beyond ~**24 layers** without stability tricks (careful warmup, etc.).

Pre-LayerNorm (GPT-3, LLaMA, …)

Layout: `x + Sublayer(LayerNorm(x))` — LayerNorm is applied **inside** the block, the residual path stays a clean **identity**.

- Gradients **flow cleanly** through the network, exactly as ResNet-style identity paths intend.
- **Stable for very deep networks** — 100+ layers without special tricks.
- The sublayer still sees normalized, well-behaved input (good optimization) while the identity path is preserved.
- **Less hyperparameter sensitivity** — robust to larger learning rates.

Why it matters (one-line intuition)

Put the normalizer *on* the highway (Post-LN) and you keep bending the through-traffic; put it *on the on-ramp* (Pre-LN) and the highway stays straight — traffic (gradients) flows for hundreds of miles (layers).

বাংলা ব্যাখ্যা: Post-LN: normalization বসে **residual পথের উপর** → gradient সংকেত বিকৃত হয়, identity copy অসম্ভব, ~24 লেয়ারের বেশি গভীর করলে অস্থির Pre-LN: normalization বসে **sublayer-এর ভেতরে**, residual পথ পরিষ্কার identity থাকে → gradient মসৃণভাবে বয়, ১০০+ লেয়ার পর্যন্ত স্থির, বড় learning rate-এও কাজ করে এজন্য GPT-3, LLaMA সব Pre-LN ব্যবহার করে

Exam-style questions

- **Explain why** Pre-LayerNorm enables much deeper Transformers than Post-LayerNorm. *(cause: Pre-LN keeps the residual path as a clean identity → mechanism: gradients flow back unmodified instead of being rotated/scaled at every block → consequence: 100+ layers train stably, whereas Post-LN destabilizes past ~24.)*
- **Application:** Your 80-layer model diverges during training with Post-LN. Name the architectural change, justify it, state a trade-off. *(change: switch to Pre-LayerNorm; justify: preserves identity residual path for clean gradient flow at depth; trade-off: Pre-LN can slightly underperform well-tuned Post-LN at shallow depths and may need final-LayerNorm adjustments.)*

MCQs

Q20. Post-LayerNorm Transformers are hard to train deeply because: A. they have too few parameters. B. normalization on the residual path distorts gradients and blocks identity mapping. ✓ C. they use RMSNorm. D. they lack attention. *Why:* LayerNorm sitting on the residual path rotates/scales gradients each block, preventing a clean copy path.

Q21. Pre-LayerNorm keeps the residual path as an identity, which mainly: A. increases vocabulary. B. lets gradients flow cleanly, enabling 100+ layers. ✓ C. removes positional encoding. D. reduces the embedding dimension. *Why:* an unmodified residual path is the ResNet trick that makes very deep stacks trainable.

Q22. Which modern model family relies on Pre-LayerNorm? A. The original 2017 Transformer. B. GPT-3 / LLaMA. ✓ C. RNN seq2seq. D. Word2Vec. *Why:* the original used Post-LN; deep modern decoders (GPT-3, LLaMA) adopted Pre-LN for stability.

2.8+ — The Training-Systems Story: FLOPs, GPUs, Precision, and Distributed Training

This is the single biggest gap in the main notes. The lecture spends ~12 slides on *how* pretraining is physically done. Expect at least one MCQ and possibly an application question here.

2.8a — FLOPs and the cost of training

A **FLOP** is one floating-point operation (add, multiply, or fused multiply-add). **FLOP/s** (or FLOPS) is operations *per second* — a property of the hardware. Transformers are dominated by dense matrix multiplications needing **billions to trillions of FLOPs per pass**. FLOPs determine how much compute is needed, how expensive training is, and how long it takes.

The 6ND rule. Training compute $\approx$ `C = 6 · N · D`, where `N` = number of parameters, `D` = total training tokens. The factor 6 splits as ~**2 per parameter for the forward pass** (compute outputs) and ~**4 per parameter for the backward pass** (compute gradients). It is an order-of-magnitude estimate, not exact.

How design choices scale FLOPs:

Parameter	Scaling of FLOPs
Model size (params)	linear
Number of layers (depth)	linear
Hidden size (width)	**quadratic** → depth is "cheaper" than width
Batch size	linear
Sequence length	**quadratic** (attention) + linear (MLP)
Heads / FFN width	linear

Worked example. A 7-billion-parameter model trained on 1.4 trillion tokens: `C = 6 × 7e9 × 1.4e12 = 5.88e22` FLOPs. On hardware delivering 1e15 FLOP/s sustained, that is `5.88e22 / 1e15 = 5.88e7` seconds $\approx$ **680 days** on one device — which is exactly why we need many GPUs (next sections).

2.8b — CPU vs. GPU (why training needs GPUs)

Aspect	CPU	GPU
Parallel units	8–64 cores (sequential-optimized)	10k+ parallel FP units
Throughput	~0.5–2 TFLOP/s	thousands of TFLOP/s
Memory bandwidth	~50–100 GB/s	multiple TB/s
Interconnect	10–30 GB/s	NVLink ~1 TB/s
Matrix-multiply HW	general	**Tensor Cores** (multiply-accumulate)

LLM training is **memory-bound** (constant weight/activation movement) and needs massive parallelism for matrix multiplications. CPUs make large-scale training **practically impossible**; GPU clusters with Tensor Cores cut training from "years on CPUs" to "days or weeks".

2.8c — Precision formats

Every value (weights, activations, gradients, optimizer states) is stored in a numeric format. The choice affects training stability (overflow/underflow), memory (bytes/value), compute speed (Tensor-Core throughput), bandwidth, and the max feasible model/batch size. **Lower precision → less memory → larger batches & longer context.**

Format	Total bits	Exponent (range) bits	Mantissa (precision) bits	Notes
FP32	32	8	23	historically used for training
FP16	16	5	10	limited range → needs **loss scaling**
BF16	16	8	7	**same range as FP32**, lower precision
INT8	8	—	—	mainly inference quantization
INT4	4	—	—	used for **QLoRA** finetuning

Key insight: BF16 combines FP32's numerical *range* (8 exponent bits) with FP16's memory/compute *efficiency* (16 bits total). Nearly all large LLMs trained since 2021 use **BF16**. FP16's small exponent range is why it needs loss scaling to avoid underflow.

2.8d — The NVIDIA H100 (the canonical training GPU)

The slide lists why the H100 is suited to LLM training: massive **Tensor-Core throughput** (raw speed); large **memory** (limits model/batch/context per GPU); extreme **memory bandwidth** (LLMs are memory-bound); high-speed **NVLink** (multi-GPU sharding); **advanced low-precision support** (fast, stable training); and a high power budget. Cost: ~**€35,000** (Nov 2025) — which is why training frontier models costs tens of millions.

2.8e — Distributed training (memorize the three parallelisms)

One GPU is not enough: a 70-billion-parameter model is ≈ **140 GB in BF16** (2 bytes × 70e9), before activations and optimizer states. Solutions:

Strategy	What is split	Each GPU holds	Key limitation
Data Parallelism (DP)	the *data* (different micro-batches)	a **full copy** of the model	every GPU must store the whole model
Tensor Parallelism (TP)	individual *weight matrices* (sharded within a layer)	a *slice* of each layer	needs synchronization (concat/sum) every layer
Pipeline Parallelism (PP)	the *layers* into stages	a *contiguous block* of layers	**pipeline bubbles** + inter-stage communication

- **3D Parallelism** = DP + TP + PP together, used for 10B–1T+ models that don't fit even when sharded. Note redundancy: in pure DP, **parameters, gradients, and optimizer states** are replicated in every DP instance — optimizer states (Adam m, v) alone are **2–4× the model size**.
- **ZeRO (Zero Redundancy Optimizer)** removes that redundancy by **sharding** the training states across GPUs instead of replicating them, freeing huge memory (larger models/batches/contexts). Works with DP/TP/PP; trade-off: **more communication**, far less memory.

2.8f — The result: a base model (LLaMA 3.1 8B)

Pretraining's output is the **base model**. Memorize this reference config:

Property	LLaMA 3.1 8B (base)
Architecture	decoder-only Transformer (causal LM)
Parameters	~8 billion
Layers (depth)	32 transformer blocks
Hidden size (width)	4096
Attention heads	32
FFN/MLP intermediate size	$\approx$ 14,336
Context length	up to 128k tokens
Training	self-supervised next-token prediction, large multilingual corpus

বাংলা ব্যাখ্যা: ট্রেনিং কম্পিউট $\approx$ **6ND** (N = প্যারামিটার, D = টোকেন; 2 forward + 4 backward) Width বাড়ানো **quadratic**, depth **linear** — তাই গভীর করা সস্তা GPU লাগে কারণ LLM **memory-bound** ও বিশাল parallel matrix-multiply দরকার (Tensor Core) Precision: **BF16** = FP32-এর range + FP16-এর কম মেমরি $\rightarrow$ ২০২১ থেকে প্রায় সব LLM এতেই ট্রেইনড এক GPU-তে ধরে না (70B $\approx$ 140 GB BF16), তাই **DP** (ডেটা ভাগ, পুরো মডেল কপি), **TP** (ওজন-ম্যাট্রিক্স ভাগ), **PP** (লেয়ার ভাগ); **ZeRO** training state শার্ড করে redundancy কমায় প্রিট্রেইনিং-এর ফল = **base model** (যেমন LLaMA 3.1 8B: 32 লেয়ার, hidden 4096, 32 head, 128k context)

Exam-style questions

- **Explain why** BF16 is preferred over FP16 for training large models. *(cause: FP16 has only 5 exponent bits $\rightarrow$ mechanism: its small dynamic range underflows/overflows easily and needs loss scaling, while BF16 keeps FP32's 8 exponent bits $\rightarrow$ consequence: BF16 trains stably at half the memory without loss-scaling hacks.)*
- **Explain why** a 70-billion-parameter model cannot train on a single GPU. *(cause: ~140 GB just for BF16 weights, plus activations and optimizer states that are 2–4$\times$ the model $\rightarrow$ mechanism: this far exceeds one GPU's memory and bandwidth $\rightarrow$ consequence: the model and/or data must be split across many GPUs (DP/TP/PP).)*
- **Application:** You can fit the model on one GPU but want a much larger effective batch across 8 GPUs. Name the strategy, justify it, state its limitation. *(strategy: data parallelism; justify: each GPU runs a full copy on different micro-batches and gradients are synchronized; limitation: every GPU must store a full model copy.)*

MCQs

Q23. The 6ND rule estimates training compute; the "6" comes from: A. 6 layers. B. ~2 FLOPs/parameter forward + ~4 FLOPs/parameter backward. ✓ C. 6 GPUs. D. 6 bytes per parameter. *Why:* forward $\approx$ 2 ops/param, backward $\approx$ 4 ops/param, summing to ~6 per parameter per token.

Q24. Doubling the model **width** (hidden size) scales attention/FFN FLOPs: A. linearly. B. quadratically. ✓ C. not at all. D. logarithmically. *Why:* matrix dimensions grow with width on both axes, so cost grows with width^2; depth, by contrast, is linear.

Q25. BF16's advantage over FP16 is that it: A. uses fewer bits. B. keeps FP32's exponent range (8 bits), avoiding loss scaling. ✓ C. has more mantissa bits. D. is an integer format. *Why:* same range as FP32 $\rightarrow$ stable training without underflow; it trades mantissa precision, not range.

Q26. Why is LLM training infeasible on CPUs? A. CPUs lack floating-point units. B. CPUs are sequential-optimized with low parallelism and bandwidth; no Tensor Cores. ✓ C. CPUs cannot store weights. D. CPUs use BF16 only. *Why:* training needs massive parallel matrix-multiply throughput and TB/s bandwidth that CPUs cannot provide.

Q27. In **tensor parallelism**, what is split across GPUs? A. The training data. B. Individual weight matrices within a layer. ✓ C. Whole layers into stages. D. The optimizer only. *Why:* TP shards weight matrices (each GPU computes a partial result that is then summed/concatenated); splitting *layers* is pipeline parallelism, splitting *data* is data parallelism.

Q28. In pure **data parallelism**, the main memory limitation is: A. pipeline bubbles. B. every GPU must store a full copy of the model. ✓ C. weight matrices are sharded. D. the tokenizer is duplicated. *Why:* DP replicates the whole model (and optimizer states) on each GPU; ZeRO addresses exactly this redundancy.

Q29. ZeRO reduces memory by: A. lowering the learning rate. B. sharding optimizer states/gradients/parameters instead of replicating them. ✓ C. removing attention. D. using FP32. *Why:* ZeRO eliminates the DP redundancy by distributing training states, at the cost of more communication.

Q30. Roughly how much GPU memory do the **weights** of a 70B model occupy in BF16? A. ~14 GB. B. ~70 GB. C. ~140 GB. ✓ D. ~560 GB. *Why:* BF16 = 2 bytes/param → 70e9 × 2 ≈ 140 GB (before activations and optimizer states).

2.10+ — The Two-Stage Paradigm and the Limitations of SFT

The main notes cover the *mechanics* of SFT, RLHF, and DPO. The slides frame *why* alignment is a separate stage and *why SFT alone is not enough* — the motivation that makes RLHF/DPO necessary.

The two-stage training paradigm

Stage 1: Pretraining	Stage 2: Post-Training (Alignment)
Acquire broad linguistic, semantic, world knowledge	Shape raw capability into deliberate, goal-directed behaviour
Learn grammar, discourse, reasoning, factual associations from large-scale text	Train the model to **interpret and follow human instructions**
Build robust representations across domains/styles	Guide outputs toward **helpful, correct, context-appropriate** responses
Outcome: a powerful **base model**, *not yet aligned*	Reduce/suppress harmful, biased, unsafe behaviour; encode human preferences
	Outcome: a model **aligned with human intentions, not just text statistics**

Why separate the stages? Pretraining is **extremely expensive → done once**; alignment is **cheap → iterated continuously**. Separation prevents catastrophic deviation from pretrained knowledge and gives clearer control over behaviour.

Limitations of SFT (why we need RLHF/DPO)

SFT teaches by **imitation** of good demonstrations. Its structural weaknesses:

1. **Imitation only / no notion of "better vs. worse".** It copies demos; it has no ranking signal.
2. **No negative feedback.** Everything in SFT is treated as "correct" — it cannot *penalize* bad or unsafe responses.
3. **No guarantee of truthfulness.** Next-token loss rewards **fluency, not accuracy** → hallucinations persist.
4. **Limited safety.** It demonstrates safe answers but cannot *punish* unsafe alternatives.
5. **Poor handling of ambiguity.** Trains only on the provided style; struggles with open-ended, multi-turn, or conflicting instructions.
6. **Style overfitting.** Tends toward an overly verbose, overly polite, generic assistant tone.

The fix: **preference-based** methods (RLHF, DPO) add the missing "better-vs-worse" signal by training on *comparisons*, so the model can be pushed *away* from bad outputs — something SFT structurally cannot do.

বাংলা ব্যাখ্যা: ট্রেনিং দুই ধাপে: (১) **Pretraining** — একবার, খুব ব্যয়বহুল, base model বানায়; (২) **Post-training/alignment** — সস্তা, বারবার করা যায়, মডেলকে মানুষের নির্দেশ মানতে শেখায় SFT শুধু **নকল (imitation)** শেখে — ভালো-খারাপ র‍্যাঙ্কিং নেই, **নেতিবাচক ফিডব্যাক নেই**, সত্যতার নিশ্চয়তা নেই (fluency শেখে, accuracy নয়) তাই **RLHF/DPO** দরকার — তারা তুলনা (comparison) থেকে "ভালো বনাম খারাপ" সংকেত যোগ করে, মডেলকে খারাপ আউটপুট থেকে **দূরে** ঠেলতে পারে

Exam-style questions

- **Explain why** supervised fine-tuning alone cannot stop a model from hallucinating. *(cause: SFT uses the next-token loss, which rewards fluent continuations → mechanism: it treats every demonstration as correct and has no signal that penalizes false statements → consequence: confident, fluent falsehoods are never punished, so hallucination persists.)*
- **Application:** After SFT your assistant is polite but sometimes confidently wrong and ignores when a user corrects it. Name the next alignment step, justify it, state a trade-off. *(step: RLHF or DPO using human preference comparisons; justify: adds a "better-vs-worse" signal that SFT lacks, penalizing bad answers; trade-off: needs preference data + (for RLHF) a reward model and an unstable RL loop, risking reward hacking / over-optimization.)*

MCQs

Q31. The two-stage paradigm separates pretraining and alignment mainly because: A. they use different architectures. B. pretraining is expensive and done once, while alignment is cheap and iterated. ✓ C. alignment must come first. D. tokenizers differ between stages. *Why:* cost asymmetry plus the desire to avoid catastrophic deviation from pretrained knowledge.

Q32. Which is a fundamental limitation of SFT? A. It updates no parameters. B. It treats all examples as correct and gives no negative feedback. ✓ C. It requires a reward model. D. It cannot use the next-token loss. *Why:* SFT is pure imitation; without comparisons it cannot penalize bad/unsafe outputs.

Q33. SFT fails to guarantee truthfulness because its loss rewards: A. accuracy. B. fluency (likely next tokens), not factual correctness. ✓ C. brevity. D. safety. *Why:* next-token likelihood $\neq$ truth, so fluent hallucinations remain unpenalized.

Q34. Preference-based methods (RLHF/DPO) add what that SFT lacks? A. A tokenizer. B. A "better-vs-worse" ranking signal from comparisons. ✓ C. Positional encoding. D. A larger vocabulary. *Why:* training on preferred-vs-rejected pairs lets the model move away from worse outputs.

2.11+ — Scaling Laws: Kaplan vs. Chinchilla, and Compute-Optimal $\neq$ Deploy-Optimal

The main notes give the Chinchilla `D` $\approx$ `20N` rule. The slides tell the fuller story — *two* competing scaling laws and a crucial production twist.

What a scaling law is

Performance (validation loss) improves in a **predictable, smooth power-law** as you increase **parameters (N)**, **training tokens (D)**, and/or **compute (C)**. This holds across many orders of magnitude. It gives **predictability** (forecast loss before training from small runs), **resource planning** (how big a model, how much data), and explains **why "scaling up" worked** from 2018–2023.

Kaplan power laws (OpenAI, 2020)

Loss follows `L = L∞ + (stuff) · N^(−α) + …`, where:

- `L∞` = **irreducible loss** = the entropy of natural language; you can never go below it.
- The exponents are **small (~0.05–0.1)** → strong **diminishing returns**, but improvement *persists* over many orders of magnitude.

Kaplan established that **"scaling works"** and launched the **"bigger is better"** era. **But** the study assumed **fixed data/training settings**, which led it to **overestimate optimal model size** and **underestimate how much data** large models need.

Chinchilla scaling laws (DeepMind, 2022)

DeepMind trained **400+ models** (70M–16B params, 5B–500B tokens) with learning-rate schedules matched to each horizon, to find the **compute-optimal** split of a fixed FLOP budget between parameters and tokens.

- **Result: parameters and data should scale *equally*.** Double the model → double the tokens. Compute-optimal ≈ **20 training tokens per parameter**.
- **Opposite of Kaplan:** no more "huge model, tiny dataset". Optimal models are **smaller and better-trained**.
- **Worked example (10× compute):** Chinchilla → increase N by ≈ 3× and D by ≈ 3× (since both scale as ~C^0.5). Kaplan would have said ≈ 5.5× params but only ≈ 1.8× data — over-investing in size.
- **Concrete correction: GPT-3 (175B)** was trained on ~300B tokens but, by Chinchilla, *should* have seen **3–4 trillion** — it was badly **undertrained**. Undertraining means higher loss for the same compute, worse downstream accuracy, and inefficient inference.

Compute-optimal ≠ Deploy-optimal (the production twist)

- **Compute-optimal** minimizes *training* loss for a fixed *training* budget (Chinchilla's ~20 tokens/param). It optimizes *training efficiency, not final quality at fixed inference cost.*
- **Deploy-optimal:** companies care about **inference cost**, which is paid **forever** (millions–billions of requests), while **training is paid once**. So it is often better to **overtrain a *smaller* model** — train it on **far more** data than compute-optimal — so it becomes much more capable at the **same inference cost**.
- **Result:** production models (LLaMA 3, Mistral, DeepSeek, Claude, GPT-class) often use **>100 training tokens per parameter** — far past the Chinchilla point — deliberately "wasting" training compute to save inference compute.

Intuition

Chinchilla asks "given my training bill, what's the best model I can train?" Deployment asks "given that I'll serve this model a billion times, what's the cheapest model that's good enough?" The second favours a small, heavily-trained model.

বাংলা ব্যাখ্যা: Kaplan (2020): loss একটা power-law-এ কমে; L∞ = ভাষার অপরিবর্তনীয় entropy (এর নিচে নামা যায় না); exponent ছোট (diminishing returns) কিন্তু Kaplan **মডেল সাইজ বেশি, ডেটা কম** বলে ভুল করেছিল **Chinchilla (2022):** প্যারামিটার ও ডেটা **সমানভাবে** বাড়াও — compute-optimal ≈ **প্রতি প্যারামিটারে ২০ টোকেন**; GPT-3 (175B) আসলে undertrained (৩০০B-র বদলে ৩-৪ trillion টোকেন দরকার ছিল) **Compute-optimal ≠ Deploy-optimal:** inference খরচ বারবার লাগে, training একবার — তাই ছোট মডেলকে **অতিরিক্ত ট্রেইন** (>১০০ টোকেন/প্যারামিটার) করাই বাস্তবে লাভজনক

Exam-style questions

- **Explain why** modern production LLMs are often trained well past the Chinchilla compute-optimal point. *(cause: inference cost is paid on every request forever while training is a one-time cost → mechanism: overtraining a smaller model makes it more capable at the same inference cost → consequence: companies "waste" training compute (>100 tokens/param) to minimize lifetime inference cost.)*
- **Application:** You have a fixed training budget and want the lowest training loss. Which scaling law guides you, and what allocation does it imply? *(law: Chinchilla; allocation: scale parameters and tokens equally, ≈ 20 tokens per parameter; trade-off: this is training-optimal, not necessarily best for cheap inference at scale.)*

MCQs

Q35. The irreducible loss L∞ in scaling laws represents: A. a bug in training. B. the entropy of natural language — a floor you cannot beat. ✓ C. the loss at initialization. D. the optimizer's learning rate. *Why:* even a perfect model faces inherent unpredictability in language; loss cannot drop below this.

Q36. Chinchilla's correction to Kaplan was that: A. bigger models are always better. B. parameters and training tokens should scale equally (~20 tokens/param). ✓ C. data does not matter. D. compute is irrelevant. *Why:* Kaplan over-weighted model size; Chinchilla showed balanced scaling and that many models were undertrained.

Q37. GPT-3 (175B) is cited as an example of: A. an overtrained model. B. an undertrained model (too few tokens for its size). ✓ C. a compute-optimal model. D. a deploy-optimal model. *Why:* ~300B tokens vs. the 3–4T Chinchilla would prescribe → undertrained.

Q38. "Compute-optimal ≠ deploy-optimal" implies that for production it is often best to: A. train the largest possible model. B. overtrain a smaller model on far more tokens. ✓ C. skip alignment. D. reduce the vocabulary. *Why:* a small, heavily-trained model minimizes the inference cost that dominates a deployed model's lifetime.

Q39. Why did Kaplan's laws overestimate optimal model size? A. They used too many models. B. They assumed fixed data/training settings, underestimating data needs. ✓ C. They ignored parameters. D. They used Chinchilla's data. *Why:* with data/schedule held fixed, the analysis credited size too much and tokens too little.

2.12+ — The Full Evaluation Taxonomy

The main notes cover perplexity and a few benchmarks. The slides present a **three-level taxonomy** plus safety. Memorize the structure — "which metric for which task" is classic MCQ and application material.

Level 1 — Intrinsic evaluation: Perplexity

Measures **predictive quality** — how well the model predicts the next token. **Lower perplexity = better.** It is directly tied to cross-entropy loss (perplexity = `exp(cross-entropy)`) and equals the model's **average uncertainty / effective branching factor**. **Limitation:** it does *not* measure reasoning, instruction-following, tool use, factuality, or safety — **a model with low perplexity can still fail real tasks.**

Level 2 — Task-level metrics (classical NLP)

Metric	Task	Measures	Intuition
Accuracy	classification (sentiment, topic)	fraction correct	simple single-label correctness
BLEU	machine translation	**n-gram precision** vs. reference + brevity penalty	"of the words I generated, how many are correct?"
ROUGE	summarization	**n-gram recall** (ROUGE-1/2) + longest common subsequence (ROUGE-L)	"of the important reference words, how many did I include?"
Pass@k	code generation	probability ≥1 of k samples passes unit tests	robustness/reliability of code

BLEU example. Reference "the cat is on the mat"; candidate "the cat sat on the". 1-gram matches = the, cat, on, the → **4 of 5**; 2-gram matches = "the cat", "on the" → **2 of 4**. Score = combine the n-gram precisions, then apply a **brevity penalty** for the too-short candidate. Scale 0–100: 30–40 decent, 50+ strong MT. **Mnemonic: BLEU = Precision (generation side).**

ROUGE example. Reference "cats sit on warm mats in the sun" (8 unigrams); candidate "cats sit on mats". ROUGE-1 = 4 of 8 = **0.50**; ROUGE-2 = 2 of 7; ROUGE-L = 4 of 8. **Mnemonic: ROUGE = Recall (reference side).**

Pass@k. With n samples and c correct, `Pass@k = 1 − C(n−c, k)/C(n, k)`. If a model solves a task 30 % of the time on the first try (Pass@1 = 0.30), then over 5 attempts there is a **91.7 %** chance at least one passes (Pass@5 ≈ 0.917).

Level 2 — Capability benchmarks (academic)

Benchmark	Tests	Format / Metric	Note
MMLU	broad knowledge + reasoning, **57 subjects**	4-choice MCQ, **accuracy**	<50 % below-undergrad, 70–85 % instruct-tuned, 95 % expert
BIG-Bench	~200 off-distribution tasks (logic, ethics, symbolic)	mixed; **no single score**	reveals **emergent abilities** at scale
HumanEval	**164** hand-written Python tasks	**Pass@k** on hidden unit tests	standard coding benchmark

Level 3 — Practical capability evaluation (what AI engineers actually care about)

Real products need more than academic scores. The slides list six dimensions, each with benchmarks — **none of these correlate well with perplexity or MMLU**:

1. **Multi-turn instruction following** — hold constraints over 5–10 turns, self-correct (MultiChallenge, COLLIE).
2. **Tool use & API-calling reliability** — schema-correct calls; *most real-world failures are tool-call mistakes, not reasoning errors* (τ^2**-bench**).
3. **Long-horizon reasoning** — multi-step, dependency-heavy planning (SWE-bench, Aider Polyglot).
4. **Factuality, hallucination & deception** — invents facts? misreports success? (FActScore, LongFact).
5. **Modality-specific** — text↔image, OCR, video (VQA, ChartQA, DocVQA, VideoMME).
6. **Domain-specific** — health/finance/law, where general models can fail catastrophically (HealthBench, FinBen, LegalEval, GSM8K/MATH).

τ^2-bench (Tau-2) — the agentic tool-use benchmark

Evaluates whether an LLM can **correctly execute multi-step tasks using external tools/APIs** across domains (Retail, Airline, Telecom). A task is a multi-turn conversation: the user gives an instruction → the model must **call the correct sequence of APIs** → the system returns intermediate results → the model **adjusts its next step** based on tool output → the final answer must reflect the tool outcomes. It tests whether a model acts like a **reliable agent, not just a text generator** (bridge to Chapter 5).

Safety evaluation

Dimension	What it checks	Example benchmarks
Toxicity	offensive/hateful/harmful language	RealToxicityPrompts, ToxiGen
Bias & Fairness	discrimination (gender, race, age)	BiasBench, CrowS-Pairs
Harmful instruction following	can it be jailbroken into harmful advice?	adversarial/jailbreak tests
Privacy & data leakage	repeats user data / memorized training text	membership inference
Hallucination safety	confident false claims causing real-world harm	factuality benchmarks

Benchmark limitations & key takeaways

- **Contamination & memorization** — benchmarks leak into training data → overestimated scores.
- **Saturation** — MMLU now >90 %, losing discriminative power.
- **Poor product indicator** — great MMLU / low perplexity can still fail on domain tasks, long instructions, precise tool calls.
- **No user-experience measure** — helpfulness, politeness, stability need human evaluation.

Key takeaways: No single number summarizes LLM quality. Evaluation must be **multi-dimensional** — combine intrinsic + task-level + capability. AI engineers must test in their **real domain, modality, and workflow**; practical-capability evaluation often predicts product success better than classical NLP metrics.

বাংলা ব্যাখ্যা: মূল্যায়ন তিন স্তরে — (১) **Intrinsic**: perplexity (next-token, cross-entropy-এর exp; কম = ভালো; কিন্তু reasoning/safety মাপে না) (২) **Task-level**: Accuracy; **BLEU = precision** (অনুবাদ, n-gram যা বানালাম তার কতটা ঠিক); **ROUGE = recall** (সারাংশ, reference-এর কতটা ধরলাম); **Pass@k** (কোড, k-বারে অন্তত একবার ইউনিট-টেস্ট পাস) আর benchmark: **MMLU** (৫৭ বিষয়, MCQ accuracy), **BIG-Bench** (~২০০ অদ্ভুত টাস্ক, emergent ability), **HumanEval** (১৬৪ Python, Pass@k) (৩) **Practical capability**: multi-turn, tool-use, long-horizon, factuality, modality, domain — এগুলো perplexity/MMLU-র সাথে মেলে না τ^2**-bench** = এজেন্ট-সুলভ tool-use (Retail/Airline/Telecom) **Safety**: toxicity, bias, jailbreak, privacy, hallucination মূল কথা: **একটা সংখ্যা দিয়ে LLM-এর মান বোঝানো যায় না** — বহুমাত্রিক মূল্যায়ন লাগে

Worked example — BLEU vs. ROUGE on the same pair

Reference: "the quick brown fox" (4 unigrams). Candidate: "the quick fox" (3 unigrams). Unigram matches = the, quick, fox = 3. - **BLEU (precision):** 3 correct / **3 generated** = **1.00** (before brevity penalty, which then lowers it for being short). - **ROUGE-1 (recall):** 3 matched / **4 in reference** = **0.75**.

Same texts, different numbers — because BLEU divides by what you *said* and ROUGE by what you *should have said*. This is the single most-tested BLEU/ROUGE distinction.

Exam-style questions

- **Explain why** a low perplexity does not guarantee a useful assistant. *(cause: perplexity only measures next-token prediction quality → mechanism: it is blind to reasoning, instruction-following, tool use, factuality, and safety → consequence: a model can predict tokens well yet fail real multi-turn, tool-using tasks.)*
- **Application:** You ship an agent that books flights via APIs and it frequently calls the wrong endpoint. Which benchmark targets this, and what does a low score tell you? *(benchmark: τ^2-bench (tool-use, e.g. Airline domain); a low score indicates unreliable multi-step tool calling — the dominant real-world failure mode — not a reasoning deficit.)*

MCQs

Q40. BLEU and ROUGE differ primarily in that: A. BLEU is for code, ROUGE for images. B. BLEU measures n-gram **precision** (translation), ROUGE measures n-gram **recall** (summarization). ✓ C. BLEU needs no reference. D. ROUGE ignores n-grams. *Why:* BLEU divides matches by generated n-grams; ROUGE divides by reference n-grams.

Q41. Perplexity is best described as: A. accuracy on MMLU. B. exponentiated cross-entropy — the model's average uncertainty per token. ✓ C. a safety metric. D. tool-call success rate. *Why:* perplexity = exp(cross-entropy), an intrinsic measure of next-token prediction; lower is better.

Q42. Pass@k is the metric of choice for: A. translation. B. summarization. C. code generation (HumanEval). ✓ D. toxicity. *Why:* it measures the probability that at least one of k generated programs passes hidden unit tests.

Q43. MMLU mainly measures: A. tool-calling reliability. B. broad multiple-choice knowledge & reasoning across 57 subjects. ✓ C. translation quality. D. summarization recall. *Why:* MMLU is 4-choice accuracy across STEM/humanities/social-science/professional subjects.

Q44. τ^2-bench (Tau-2) evaluates whether a model: A. predicts the next token well. B. acts like a reliable agent, correctly calling sequences of tools/APIs. ✓ C. translates between languages. D. minimizes perplexity. *Why:* it tests multi-step tool use across domains like Retail/Airline/Telecom — agentic reliability.

Q45. Which is a stated limitation of academic benchmarks? A. They are too small to run. B. Contamination and saturation can make scores overestimate real performance. ✓ C. They measure user experience directly. D. They require BF16. *Why:* leaked test data inflates scores and saturated benchmarks (MMLU >90 %) lose discriminative power.

Q46. "No single number summarizes LLM quality" implies evaluation should be: A. based on perplexity alone. B. multi-dimensional (intrinsic + task-level + capability + safety). ✓ C. done once before release. D. limited to MMLU. *Why:* different metrics capture different, weakly-correlated aspects of quality.

Q47. A model with great MMLU but failing your production workflow most likely needs: A. a smaller vocabulary. B. practical-capability evaluation in your real domain/modality/workflow. ✓ C. lower precision. D. a new tokenizer. *Why:* academic scores poorly predict product success; test the actual deployment conditions.

Extended-Topics Cheat Sheet (night-before-exam, one line each)

Topic	The one thing to remember
Data bias	Model learns **correlations, not truth**; training **amplifies** mild data asymmetries
Language bias	GPT-4 best in English because English is **overrepresented** (same model, data artefact)
Vocab granularity	char = no OOV but long; **subword = best balance**; word = short but huge vocab + OOV
Vocab size	small vocab → longer sequences → more compute; large vocab → more embedding memory
Tokenizer + model	**inseparable** — cannot swap tokenizer after training
Context window	max input+output tokens; **fixed architectural** property; overflow → drop/summary/RAG
One-hot limits	all **orthogonal** (no relation), dimensionality explosion, no generalization
Embedding layer	a **linear projection** of one-hot; meaning lives in **distances & directions**
Norm vs direction	**norm ≈ frequency/generality, direction ≈ meaning** → use cosine
Word2Vec	**skip-gram**: predict context from center word; static (no context)
Attention advantages	kills info **bottleneck**, **shortcut** gradients, interpretable **attention maps**
Semantic distortion	positional encoding only slightly shifts direction (embedding magnitude ≫ positional)
Post-LayerNorm	norm on residual path → unstable past ~**24 layers**
Pre-LayerNorm	identity residual path → **100+ layers** (GPT-3, LLaMA)
FLOPs (6ND)	2 forward + 4 backward per param·token; **width quadratic, depth linear**
CPU vs GPU	training is **memory-bound** + massive parallel matmul → needs GPU **Tensor Cores**
Precision	**BF16** = FP32 range + FP16 memory; standard since 2021; FP16 needs **loss scaling**
H100	Tensor Cores + huge bandwidth + NVLink; ~€35k each
DP / TP / PP	split **data / weight-matrices / layers**; **ZeRO** shards states to cut DP redundancy
70B in BF16	≈ **140 GB** of weights → can't fit on one GPU
Base model	output of pretraining (e.g. LLaMA 3.1 8B: 32 layers, hidden 4096, 32 heads, 128k ctx)
Two-stage	pretrain (expensive, once) → align (cheap, iterated)
SFT limits	**imitation only, no negative feedback, fluency≠truth** → needs RLHF/DPO

Topic	The one thing to remember
Kaplan (2020)	"scaling works", but overestimated size, underestimated data
Chinchilla (2022)	scale N and D **equally**, ≈ **20 tokens/param**; GPT-3 was undertrained
Compute≠Deploy	inference paid forever → **overtrain small models** (>100 tokens/param)
Perplexity	exp(cross-entropy); intrinsic only — blind to reasoning/safety
BLEU vs ROUGE	**BLEU = precision** (translation); **ROUGE = recall** (summarization)
Pass@k	code: P(≥1 of k samples passes unit tests); Pass@1=0.30 → Pass@5≈0.917
MMLU / BIG-Bench / HumanEval	57-subject MCQ / ~200 off-distribution tasks / 164 Python tasks
τ^2-bench	agentic **tool-use** reliability (Retail/Airline/Telecom)
Eval takeaway	**no single number**; evaluate multi-dimensionally in your real workflow

বাংলা ব্যাখ্যা: উপরের টেবিলটাই পরীক্ষার আগের রাতের দ্রুত-রিভিশন প্রতিটা সারি একেকটা সম্ভাব্য MCQ বা "Explain why" প্রশ্নের বীজ — ডান কলামের এক লাইন মুখস্থ থাকলেই অর্ধেক উত্তর হয়ে যায়

Self-test answer key (Q1–Q47)

1-B · 2-B · 3-B · 4-B · 5-B · 6-B · 7-B · 8-B · 9-D · 10-B · 11-B · 12-B · 13-C · 14-B · 15-B · 16-B · 17-B · 18-B · 19-B · 20-B · 21-B · 22-B · 23-B · 24-B · 25-B · 26-B · 27-B · 28-B · 29-B · 30-C · 31-B · 32-B · 33-B · 34-B · 35-B · 36-B · 37-B · 38-B · 39-B · 40-B · 41-B · 42-C · 43-B · 44-B · 45-B · 46-B · 47-B

> The 47 MCQs above test **only** the extended topics. For 50+ intuitive MCQs covering **all** of Chapter 2 (original + extended), see `Chapter_02_MCQs`. The correct-answer letters here are intentionally varied in the full bank — don't pattern-match "mostly B"; that is an artefact of how options were ordered in this supplement, and is fixed in the dedicated MCQ bank.

End of Chapter 2 Extended supplement.