Tokenization & Pretraining Paradigms

Lecture 14 · ES 667: Deep Learning

Prof. Nipun Batra
IIT Gandhinagar · Aug 2026

Where we are

Last lecture: the **Transformer block**. Stack it, add positional encoding, mask if autoregressive.

But stacking requires *inputs*. And inputs are **discrete symbols** (characters, subwords, words), not vectors.

> REFERENCE
>
> Today maps to **Prince Ch 12 (pretraining)** + Karpathy's *Let's build the GPT Tokenizer* video. Tokenization is the part of LLMs everyone wants to skip — don't.

Four questions:

1. Why is tokenization hard?
2. How does **BPE** work step-by-step?
3. What are the three **pretraining paradigms** — BERT, GPT, T5?
4. Why does tokenization cause so many LLM bugs?

PART 1

Why tokenization is hard

Spoiler · there's no good answer

Learning outcomes

By the end of this lecture you will be able to:

1. Explain why **tokenization** is the "first design decision of an LLM."
2. Run the **BPE** algorithm on paper for 5 merges.
3. Distinguish **character / word / subword / byte-BPE** and pick appropriately.
4. Identify common **tokenization bugs** (counting letters, arithmetic, whitespace).
5. Contrast the three **pretraining paradigms** · BERT · GPT · T5.
6. Name the **data + compute economics** of a 70B-param training run.

Three failed alternatives

UNIT	PROBLEM
Characters	sequences are 5–10× longer → attention becomes expensive ($O(N^2)$)
Whole words	vocab must be ~10^6; unseen words → OOV; misspellings fail
Morphemes	language-specific; requires linguistic annotation; doesn't scale

KEY IDEA

Subword tokenization is the compromise: common words as single tokens, rare words split into learned subwords.

A concrete failure · character LMs

Train a character-level Transformer on Wikipedia. It *works* — GPT-like samples of readable English.

But it's expensive:

- A 1000-word page → ~6000 characters → 6000 attention positions → **36M attention entries per layer**.
- Equivalent word-level model: 1000 tokens → **1M entries**. 36× cheaper.

Plus · character LMs have to *learn* that "t-h-e" is a word, unit by unit. That's capacity wasted on a solved problem.

> INTUITION
>
> Subwords compromise: keep common sequences as one unit (saving sequence length), split rare sequences (keeping open vocabulary). The winning middle path.

Smart-dictionary analogy

KEY IDEA

A tokenizer builds a **dictionary** for the language.

- Character dictionary · just the alphabet · too basic.
- Word dictionary · huge · breaks on "un-un-believable".
- **Subword dictionary** · common words plus reusable prefixes (`un-`) and suffixes (`-able`) · can compose any new word.

That's the sweet spot we'll build with BPE on the next slides.

The sweet spot · subwords

Ideal subword tokenizer:

- Common words → single tokens (cheap, frequent)
- Rare words → composition of familiar subwords (generalizable)
- **No OOV** — any byte sequence is tokenizable
- Vocab size tunable (typically 30k–100k)

The winning algorithm: **Byte-Pair Encoding** (BPE), re-purposed from 1994 data compression.

PART 2

BPE step-by-step

One merge rule at a time

BPE merges · visual

BPE · greedy merges of the most frequent adjacent pair

STEP 0 · START FROM CHARACTERS

corpus · "lowered loudly"

| l | o | w | e | r | e | d | · | l | o | u | d | l | y |

STEP 1 · MOST FREQUENT PAIR = (L, O) · MERGE → "LO"

| lo | w | e | r | e | d | · | lo | u | d | l | y |

STEP 2 · NEXT MOST FREQUENT = (L, Y) · MERGE → "LY"

| lo | w | e | r | e | d | · | lo | u | d | ly |

STEP 3 · NEXT = (E, D) · MERGE → "ED" · PICKS UP PAST-TENSE SUFFIX

| lo | w | e | r | ed | · | lo | u | d | ly |

AFTER K MERGES (K = TYPICAL 30,000 FOR GPT-2)

| lowered | · | loudly |

KEY PROPERTIES

- Starts from individual characters (or bytes)
- Greedy merge of most frequent adjacent pair
- Stops at target vocab size (typically 30k–100k)
- Frequent words become one token ("the", "is")
- Rare words split into subwords ("untreatable" → "un" + "treat" + "able")
- **No OOV** — any new word tokenizable as bytes

IN PRACTICE

Interactive: type a corpus, press "merge" to see the most frequent pair get glued live — bpe-merges.

The BPE algorithm · 7 lines

```python
def train_bpe(corpus, n_merges):
    # 1. Split every word into characters
    tokens = [list(word) for word in corpus.split()]
    merges = []

    for _ in range(n_merges):
        # 2. Count all adjacent pairs
        pair_counts = Counter((a, b) for word in tokens for a, b in zip(word[:-1], word[1:]))
        if not pair_counts: break

        # 3. Find the most frequent pair
        best = pair_counts.most_common(1)[0][0]
        merges.append(best)

        # 4. Apply the merge across all tokens
        tokens = [merge_in_word(w, best) for w in tokens]

    return merges, tokens
```

At inference, apply the same merge rules in order → tokenize any new string.

Worked BPE · merge trace

BPE merge trace · "low lower newest widest"

step	tokens	merge rule · freq
0	l o w · l o w e r · n e w e s t · w i d e s t	initial char-level split
1	l o w · l o w e r · n e w **es** t · w i d **es** t	(e, s) → **es** · freq=2
2	l o w · l o w e r · n e w **est** · w i d **est**	(es, t) → **est** · freq=2
3	**lo** w · **lo** w e r · n e w est · w i d est	(l, o) → **lo** · freq=2
4	**low** · **low** e r · n e w est · w i d est	(lo, w) → **low** · freq=2
5	low · low e r · n **we** est · w i d est	(w, e) → **we** · freq=1

Learned vocabulary (5 merges applied)

l · o · w · e · r · n · s · t · i · d · es · est · lo · low · we

Common words like "low" are now single tokens. Rare words still split into subwords. No <unk> needed.

At inference · apply merges 1→5 in order to any new word.

BPE · the data-compression analogy

INTUITION

Like a basic file compressor. If "ABCABCABC" appears a lot, define a new symbol $Z = $ "ABC" and rewrite as "ZZZ". BPE does the same for language · find the most common adjacent pair (like `th`), compress it into a single new token, repeat.

Worked BPE · "low lower newest widest"

Step 0 · split each word into characters with `</w>` end marker.

`l o w </w>`, `l o w e r </w>`, `n e w e s t </w>`, `w i d e s t </w>`

Step 1 · count adjacent pairs.

`(l, o): 2`, `(o, w): 2`, `(e, s): 2`, `(s, t): 2`, `(w, e): 1`, `(e, r): 1`, …

Tie among `(l,o), (o,w), (e,s), (s,t)`. Pick `(e, s)`.

Merge 1 · `(e, s) → "es"`.

`l o w`, `l o w e r`, `n e w es t`, `w i d es t`

Step 2 · recount. Now `(es, t): 2` is a *new* pair tied with `(l,o), (o,w)`. Pick `(es, t)`.

Merge 2 · `(es, t) → "est"`.

`l o w`, `l o w e r`, `n e w est`, `w i d est`

Merge 3 · `(l, o) → "lo"`. Merge 4 · `(lo, w) → "low"`.

After 4 merges we have learned `low`, `est` as single tokens. At inference, apply merges 1–4 in order to any new word.

Worked BPE · second example

Corpus · `"hug bug rug"`. Initial: `h u g </w>`, `b u g </w>`, `r u g </w>`.

Count pairs. $(u, g) : 3$, $(g, \langle/w\rangle) : 3$, (h, u), (b, u), (r, u) each 1.

Merge 1 · $(u, g) \to$ **"ug".** New: `h ug </w>`, `b ug </w>`, `r ug </w>`.

Recount. $(ug, \langle/w\rangle) : 3$, others 1.

Merge 2 · $(ug, \langle/w\rangle) \to$ **"ug</w>".** Final: `h ug</w>`, `b ug</w>`, `r ug</w>`.

The algorithm has discovered the reusable suffix `ug` and the morphologically meaningful unit `ug</w>`.

Why byte-level BPE is the default

Two breakthroughs GPT-2 introduced:

1. **Start from bytes (0–255), not Unicode characters.** Every possible string becomes tokenizable, including emojis, foreign scripts, binary garbage.
2. **Pretokenize by regex** before BPE, to avoid crossing word boundaries ("New York" stays as two separate merge chains).

> KEY IDEA
>
> Result · a 50k-token vocab that covers English, code, Japanese, emoji, and anything else users throw at it. No `<unk>` token needed.

Llama, GPT-*, Mistral, Claude all use byte-level BPE with minor tweaks. SentencePiece is the same idea packaged for cross-language training.

Tokenizer comparison · same sentence, different counts

Same text · four tokenizers · very different counts

`Input · "The unbelievable complexity of modern tokenizers!"`

Char-level

52 tokens · one per character

Word-level

7 tokens · one per whitespace-separated word · huge vocab · OOV risk

The	unbelievable	complexity	of	modern	tokenizers	!

BPE (GPT-4 tokenizer)

11 tokens · subword splitting; rare words decomposed

The	un	believ	able	complexity	of	modern	token	izer	s	!

Byte-level BPE (Llama 3)

9 tokens · similar to BPE but bytes as base · handles any language / emoji / binary

Three BPE variants you will meet

VARIANT	HOW	USED IN
Character-level BPE	start from Unicode chars	original 2015 paper
Byte-level BPE	start from raw bytes	GPT-2, Llama, most modern LLMs
WordPiece	same idea, likelihood-based merge	BERT, DistilBERT
SentencePiece	treat whitespace as regular char	Llama, mT5, multilingual

INTUITION

Byte-level BPE (GPT-2) is now the default · handles any unicode, any language, any emoji, no OOV.

Tokenization gotchas · real LLM failures

WATCH OUT

"How many r's in strawberry?" — GPT-4 famously miscounted. Why? "strawberry" tokenizes to something like ["straw", "berry"] or ["str", "aw", "berry"]. The model never sees individual letters — it sees chunks.

Arithmetic errors. Numbers tokenize inconsistently: "1234" might be one token, "1235" might split. Models learn arithmetic by memorizing token patterns, not digit manipulation.

Spaces matter. " the" (with leading space) is a different token from "the". This is why prompts to LLMs are sensitive to trailing spaces.

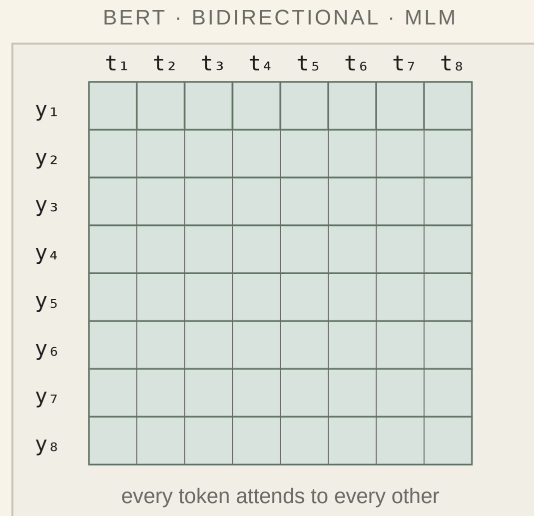
PART 3

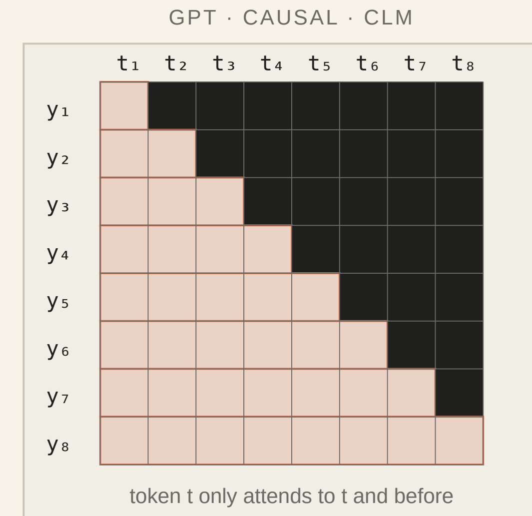
Three pretraining paradigms

Same Transformer · different objectives

Three families · one architecture

Attention masks · BERT sees everything, GPT sees only the past





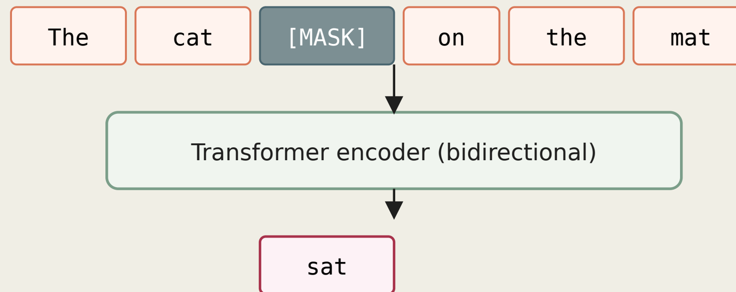
BERT *predicts masked-out tokens with full context (MLM).* ***GPT*** *predicts the next token given only the past (CLM). Same Transformer; different masks → different objectives.*

BERT vs GPT · side-by-side

MLM (BERT) vs Causal LM (GPT) · same architecture, different training task

BERT · Masked Language Modeling

mask 15% · predict masked given both sides



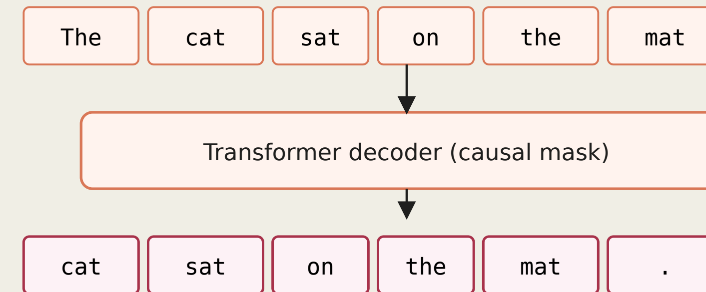
predict "sat" given "The cat __ on the mat"
loss · CE only at masked position

Properties

- ✓ bidirectional context (cheating ok)
- ✓ great for classification / retrieval
- ✗ can't generate autoregressively
- ✗ only 15% of tokens give gradient

GPT · Causal Language Modeling

predict every next token given past only



loss · CE at every position · 100% tokens teach

Properties

- ✓ natural autoregressive generation
- ✓ ALL tokens give gradient signal
- ✓ scales beautifully (GPT-3, Llama, Claude)
- ✗ can't peek ahead (can't cheat)

BERT · the cloze-test analogy

KEY IDEA

BERT plays a **fill-in-the-blanks** game · like a cloze test in school.

We hide ~15% of the words; BERT must guess them. Because it can see text **on both sides** of the blank, it gets very good at *understanding* context.

This makes BERT a strong **encoder** · ideal for tasks where you need a representation of the whole sentence (classification, retrieval, NER). It's bad at *generating* text · because it never practices producing tokens one-by-one.

BERT · the cloze-test math, step by step

1. **Sentence.** $x = [\text{The}, \text{cat}, \text{sat}, \text{on}, \text{the}, \text{mat}]$.
2. **Mask token 4.** $x' = [\text{The}, \text{cat}, \text{sat}, [\text{MASK}], \text{the}, \text{mat}]$.
3. **Goal.** Predict the original token from context. We want to maximize $P(x_4 = \text{"on"} \mid x_{\setminus 4})$.
4. **Loss.** Standard trick · negative log probability:

$$\text{loss}_4 = -\log P(x_4 \mid x_{\setminus 4})$$

5. **Total.** Sum over the ~15% masked positions:

$$\mathcal{L}_{\text{MLM}} = \sum_{i \in \text{masked}} -\log P(x_i \mid x_{\setminus i})$$

The model sees the whole sentence (no causal mask) → rich bidirectional context.

Worked numeric · BERT loss for one mask

Input · `…sat [MASK] the…`. Correct answer · "on".

Transformer outputs logits over vocab at the `[MASK]` position:

- $\mathrm{logit}(\mathrm{on}) = 3.5$, $\mathrm{logit}(\mathrm{above}) = 2.1$, $\mathrm{logit}(\mathrm{under}) = 1.5$, ...
  Softmax · $\exp(3.5) \approx 33.1$, $\exp(2.1) \approx 8.2$, $\exp(1.5) \approx 4.5$, ...
  $P(\mathrm{on}) \approx 33.1/(33.1 + 8.2 + 4.5 + \ldots) \approx \mathbf{0.75}$.

Loss · $-\log 0.75 \approx \mathbf{0.287}$.

If the model were more confident ($P = 0.99$), loss would be $-\log 0.99 \approx 0.01$ — model rewarded for confidence on the right answer.

Great for: classification, NER, retrieval (embeddings).
Bad for: generation — can't autoregressively extend.

BERT · why mask 15%?

- Mask **too few** · most sequences see no loss signal → slow training.
- Mask **too many** · target is too hard, context too sparse.

15% was found empirically. Of those 15%:

- 80% actually replaced by `[MASK]`
- 10% replaced by a random token (adds noise, helps robustness)
- 10% left unchanged (so the model can't cheat by ignoring unmasked positions)

> INTUITION
>
> This mask-then-reconstruct recipe is the same idea as the denoising autoencoder from L19 — BERT is essentially a denoising autoencoder over language, using a Transformer encoder as the denoiser.

GPT · the smartphone-keyboard analogy

INTUITION

Predictive text on your phone. Type *"I am heading to the..."* — it suggests "gym", "store", "movies". Predicts the **next word** from what you've already typed; never sees the future.

GPT does this for every word, learning to be an excellent generator.

GPT · CLM math, step by step

Sentence · $x = [\text{The}, \text{cat}, \text{sat}]$. Break into prediction problems:

1. Given `<s>`, predict "The". Loss · $-\log P(\text{The})$.
2. Given "The", predict "cat". Loss · $-\log P(\text{cat} \mid \text{The})$.
3. Given "The cat", predict "sat". Loss · $-\log P(\text{sat} \mid \text{The}, \text{cat})$.

Total · sum over all positions:

$$\mathcal{L}_{\text{CLM}} = \sum_{t=1}^{T} -\log P(x_t \mid x_{<t})$$

Causal attention mask · model can only look backward.

Worked numeric · GPT loss at one step

Context · "The cat ...". Correct next word · "sat".

Logits over vocab · $\text{logit}(\text{sat}) = 4.0$, $\text{logit}(\text{ran}) = 2.5$, $\text{logit}(\text{jumped}) = 2.0$, ...

Softmax · $\exp(4.0) \approx 54.6$, $\exp(2.5) \approx 12.2$, $\exp(2.0) \approx 7.4$, ...
$P(\text{sat} \mid \text{The cat}) \approx 54.6/(54.6 + 12.2 + 7.4 + \ldots) \approx \mathbf{0.78}$.

Loss at $t = 3 \cdot -\log 0.78 \approx \mathbf{0.248}$.

Total sentence loss · sum these up across every position. A 2048-token window gives 2048 little training problems for free, every step.

Great for: generation, chat, code, anything where you produce text one token at a time.
Bad for: bidirectional understanding (but at scale, GPT-3+ closed this gap).

GPT · why causal loss is so rich

One tiny objective — predict the next token — forces the model to reason about:

- **Syntax** · closing brackets, matching tenses, agreement
- **Semantics** · what words follow each other meaningfully
- **World knowledge** · "The capital of France is ..." requires a fact
- **Reasoning** · "If all A are B, and x is A, then x is ..." requires logic
- **Style** · code continuations, formal-register, poetry

KEY IDEA

Every position in a 2048-token window is a little training example. A 1T-token corpus gives you 10^{12} supervised tasks for free — no human labeling needed.

This scale-and-generality combo is why *next-token prediction* — despite looking trivial — ended up subsuming most of NLP.

T5 · encoder-decoder · text-to-text

Frame every task as text-to-text:

```
"translate English to German: The house is wonderful."
   → "Das Haus ist wunderbar."

"summarize: ‹paragraph›"
   → ‹summary›

"question: ‹q›  context: ‹c›"
   → ‹answer›
```

> REFERENCE
>
> Raffel et al. 2019 · T5 — *Text-to-Text Transfer Transformer*. Unified framework; same model does translation, summarization, QA, classification.

Survives in some translation pipelines. But for pure generation, decoder-only (GPT pattern) won.

Scaling · which paradigm won?

YEAR	WINNER	WHY
2018	BERT	cheap, works, encoder embeddings useful
2020	GPT-3	scale unlocked few-shot learning
2022	GPT-3.5 / InstructGPT	alignment via instruction tuning + RLHF
2023	GPT-4, Llama 2	decoder-only becomes the dominant paradigm
2026	decoder-only + tool use + reasoning	everyone converges here

IN PRACTICE

Decoder-only won the LLM race. BERT still ships in retrieval pipelines (small, fast, good embeddings). T5 survives where structured I/O matters.

PART 4

What pretraining actually learns

The "foundation" in foundation model

Why pretraining works so well

Predicting the next token on a trillion-token corpus forces the model to learn:

- **Grammar** — what syntactic constructions are valid
- **Facts** — what follows "The capital of France is"
- **Reasoning patterns** — what tokens usually complete "if A then"
- **Style** — code, poetry, legal writing, instructions

KEY IDEA

Next-token prediction is so rich a task that a model good at it ends up learning **most of what's in the data** — implicitly, without any labeled supervision.

Pretraining data · where does 15T tokens come from?

Web-scale sources

- **Common Crawl** · ~2T web pages (filtered)
- **C4** · curated Common Crawl · 750B tokens
- **RefinedWeb** · quality-filtered · 5T tokens
- **Wikipedia** · 10B tokens (20 languages)

Specialized

- **GitHub code** · 1-2T tokens (de-licensed)
- **ArXiv + books3 + StackExchange** · 100B+
- **Synthetic data** (Phi-3) · textbooks generated by stronger models
- **Multilingual scraped** · CCMatrix, mC4

WATCH OUT

Copyright issues · training on copyrighted text without license is legally contested. NY Times v. OpenAI (2023) · unresolved. The data pipeline is as much a legal project as a technical one.

Data quality beats quantity

Phi-3 (2024) · trained on **3T** "textbook-quality" tokens (heavily filtered + synthetic); matched Llama-2 7B trained on **2T** web tokens.

DERIVATION

RECIPE	TOKENS	QUALITY	OUTCOME
15T web (Llama-3)	large, noisy	baseline	strong LLM
3T curated (Phi-3)	small, high	synth	matches a larger model
1T books (Books3)	medium, high	literary	great for story writing

KEY IDEA

Data curation is now the hottest research area. Dedup, language-ID, toxicity filter, repetition filter, perplexity filter with smaller model, synthetic augmentation. Each step buys 1-5 points on benchmarks.

Compute economics · where the 6 comes from

The famous LLM rule of thumb: **training FLOPs** $\approx 6 \cdot N \cdot D$ where N = parameters, D = training tokens.

Why 6? Per token:

1. **Forward pass.** Most compute is the FFN's two big matmuls. ≈ $\mathbf{2N}$ FLOPs per token.
2. **Backward pass.** Standard rule · ~2× the forward cost. ≈ $\mathbf{4N}$ FLOPs per token.
3. **Total per token** · $2N + 4N = \mathbf{6N}$.

Multiply by all D tokens in the dataset:

$$\text{FLOPs} \approx 6 \cdot N \cdot D$$

Worked numeric · 70B model training cost

One training run of a 70B-parameter model on Chinchilla-optimal $D = 1.4$T tokens:

- FLOPs $\approx 6 \cdot (7 \cdot 10^{10}) \cdot (1.4 \cdot 10^{12}) \approx 5.9 \times 10^{23}$
- Time · ~25 days on 4k A100 GPUs (at ~150 TFLOPS / GPU effective utilization)
- Cost · ~$5M at commercial GPU-hour rates

Smaller check · Llama 2 7B. $N = 7 \times 10^9$, $D = 2 \times 10^{12}$:
$\text{FLOPs} \approx 6 \cdot 7 \cdot 10^9 \cdot 2 \cdot 10^{12} = 8.4 \times 10^{22}$.

Llama 3 70B · reportedly ~$80M including experiments. GPT-4 class · ~$100M+ per training run.

> IN PRACTICE
>
> 10 years ago a deep net cost tens of dollars to train. Today, frontier model training costs a house.

Scaling recap · 5 years of LLMs in one chart

YEAR	MODEL	PARAMS	TOKENS	NOTABLE
2018	BERT-base	110M	3.3B	first pretrained Transformer in production
2019	GPT-2	1.5B	40B	"too dangerous to release"
2020	GPT-3	175B	300B	first few-shot emergence
2022	Chinchilla	70B	1.4T	train-compute optimal
2023	Llama 2 70B	70B	2T	open weights, Chinchilla-ish
2024	Llama 3 8B	8B	15T	aggressively over-trained for inference
2026	frontier LLMs	1T+	10T+	multi-modal, reasoning, tool use

INTUITION

Architecture has barely changed (decoder-only Transformer). What scaled: compute, data, careful engineering. "Attention + scale" really was the thing. L15 goes into the mechanical details.

Fine-tuning · from pretrained to useful

Pretrained models are knowledgeable but not steerable. To make them follow instructions, chat, or specialize on a task:

1. **Supervised fine-tuning (SFT)** on instruction-response pairs.
2. **LoRA / QLoRA** — parameter-efficient fine-tuning (next lecture).
3. **RLHF / DPO** — align with human preferences (L16).

The pretrained model is the brain. Fine-tuning is how you train it to do what you want.

Summary · Lecture 14 — summary

- **Tokenization** balances sequence length vs vocab size. Subword wins.
- **BPE** · greedy adjacent-pair merges; byte-level variant (GPT-2, Llama) has no OOV.
- **Tokenization bugs** · spelling errors, arithmetic weirdness, space sensitivity — all trace to token boundaries.
- **BERT** · encoder-only · bidirectional MLM · classification + embeddings.
- **GPT** · decoder-only · causal LM · generation. **The winner for scaled LLMs.**
- **T5** · encoder-decoder · text-to-text framing. Niche role in 2026.

Read before Lecture 15

Chinchilla paper (Hoffmann 2022) + HuggingFace course Chapter 1.

Next lecture

Large Language Models — Chinchilla scaling, RoPE, GQA, distributed training, emergent abilities.

> NOTEBOOK
>
> **Notebook 14** · `14-bpe-from-scratch.ipynb` — implement BPE tokenizer from scratch; train on a small corpus; visualize merges; tokenize new sentences.