ATPESC

Argonne NATIONAL LABORATORY | ALCF

Introduction to Large Language Models

Jingyan (Jane) Jiang

Argonne Leadership Computing Facility
Argonne National Laboratory

ATPESC 2026 | August 3, 2026

What we'll cover — and the vocabulary you'll take away

1. What *is* a large language model?
2. How does text become model input?
   tokenization, **embeddings**
3. How does it compute?
   the **Transformer** and **attention**
4. How is it built?
   training: pre-training + fine-tuning
5. Where does it fail, and how do we fix it?
   hallucination, **RAG**

One question to keep in view

How does a pile of text become something that answers your questions?

Intuition first — the mathematics behind each term is in the backup slides for Q&A.

What is a large language model?

Definition

A **large language model** (**LLM**) is a **language model** — a program that predicts language — trained on a vast amount of text. Its behavior lives in billions of learned numbers called **parameters** (or **weights**).

"Large" has no fixed cutoff: the training data, the number of parameters, and the computing power are all enormous — and keep growing.

Sources: Radford et al. (2019). *Language Models are Unsupervised Multitask Learners*. OpenAI technical report

What can it do for your science?

Read & summarize

Digest papers, docs, and long reports; extract key points.

Write & explain code

Draft scripts, translate between languages, explain errors.

Draft & edit

Papers, proposals, and documentation; rephrase and polish.

Answer & reason

Explain concepts, brainstorm, work through multi-step problems.

Connect to tools, data & skills

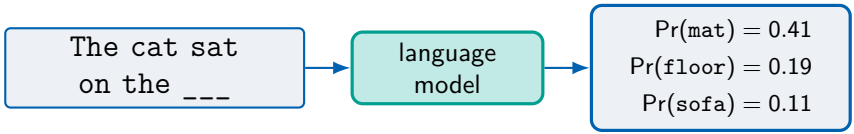
With retrieval and tools it can use *your* documents, databases, and software — and **skills** distilled from your own experience, knowledge, and insights — e.g. to help run jobs on ALCF systems.

*Using a finished model is called **inference**. Quality depends on the model, the prompt, the data, the skills, and the tools it can see and use.*

Sources: Bommasani et al. (2021). *On the Opportunities and Risks of Foundation Models.* arXiv:2108.07258

The core task: predicting the next token

"LANGUAGE MODELING"



One idea powers everything an LLM does

An LLM assigns a probability to each possible next **token** (a word or word-piece) — this is **next-token prediction**, also called **language modeling**. To write, it samples a token and feeds it back in, again and again (**autoregressive** generation).

Sources: Radford et al. (2019). *Language Models are Unsupervised Multitask Learners*. OpenAI technical report

Why predicting tokens captures knowledge

LANGUAGE MODELING AS A DISTRIBUTION OVER LANGUAGE

Language and knowledge are deeply linked

"The limits of my language mean the limits of my world."

— Ludwig Wittgenstein, *Tractatus Logico-Philosophicus* (5.6)

To predict well, it must learn facts

`Water boils at` → `100°C`

`The capital of France is` → `Paris`

`The opposite of hot is` → `cold`

The useful distinction

Learning the **probability distribution** over language forces the model to absorb patterns of knowledge in text. But it learned *language about* the world — not the world itself, so it can still be wrong.

Sources: Ludwig Wittgenstein (1922). *Tractatus Logico-Philosophicus*, proposition 5.6; Bommasani et al. (2021). *On the Opportunities and Risks of Foundation Models*. arXiv:2108.07258

Two ways to build language ability: write rules, or learn from data

WHY LLMS ARE "DATA-DRIVEN"

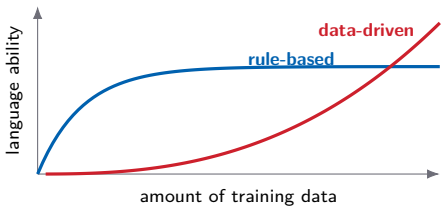
Rule-based (the old way)

People hand-write grammar and logic rules. Predictable, but brittle — real language has endless exceptions.

```
if tense == past
        ↓
use "sat", not "sit"
```

Data-driven (how LLMs work)

Show the model enormous text and let it *learn* the patterns (the distribution) itself — no rules written by hand.

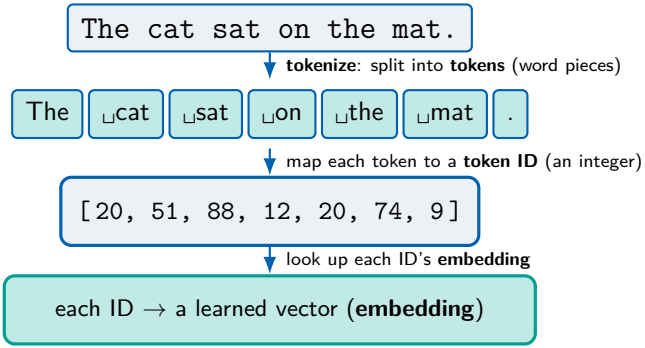


Schematic (not plotted from one dataset): hand-built rules tend to plateau while data-driven performance keeps rising with more data — a long-observed pattern.

Sources: Halevy, Norvig & Pereira (2009). *The Unreasonable Effectiveness of Data.* IEEE Intelligent Systems 24(2):8–12; Bengio et al. (2003). *A Neural Probabilistic Language Model.* JMLR 3:1137–1155

Tokenization: from text to tokens to numbers

PIPELINE STEP 1 — TURNING TEXT INTO MODEL INPUT



Tokens are the units the model reads and predicts

Tokenization splits text into **tokens**; each maps to a **token ID**, and each ID to a learned **embedding** — the form the model computes on. ("the" appears twice, so it reuses ID 20.)

Sources: Sennrich et al. (2016). *Neural Machine Translation of Rare Words with Subword Units.* ACL; arXiv:1508.07909; Kudo & Richardson (2018). *SentencePiece: A Simple and Language Independent Subword Tokenizer and Detokenizer for Neural Text Processing.* EMNLP; arXiv:1808.06226

The Transformer architecture

PIPELINE STEP 2 — THE MODEL, FROM "ATTENTION IS ALL YOU NEED" (2017)

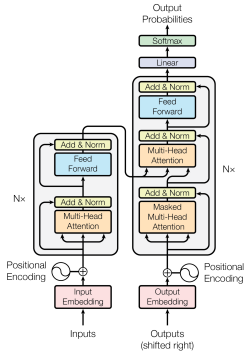


Figure 1: encoder (left) + decoder (right).

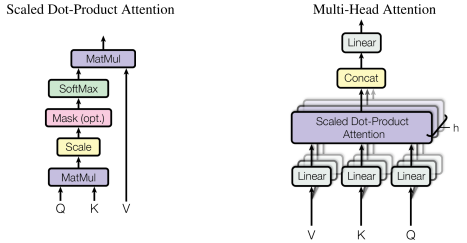


Figure 2: (scaled dot-product & multi-head) attention.

Stacked blocks of attention + feed-forward

Embeddings flow through repeated **Transformer blocks**: each mixes information across tokens with **attention**, then transforms each token with a small network. Modern LLMs use the **decoder** (right) stack.

Original figures, Vaswani et al. (2017). We build the intuition for attention next.

Sources: Vaswani et al. (2017). *Attention Is All You Need.* NeurIPS; arXiv:1706.03762

Transformer families: encoder-only, decoder-only, encoder–decoder

WHICH CONTEXT IS VISIBLE — AND WHY IT MATTERS

Encoder-only (BERT)

Sees the **whole input, both directions**. Builds a rich understanding of given text → great for **classification** and extraction (you analyze text, not generate it).

Decoder-only (GPT)

Sees only the **left context** (causal) and predicts the next token → naturally **generates** text left-to-right. The design behind modern chat LLMs.

Encoder–decoder (T5)

Encoder reads a **source** sequence; decoder writes a **target** sequence. Built for mapping one sequence to another.

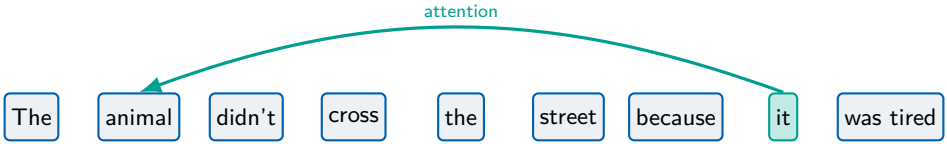
"source → target" and MT

MT = *machine translation*, the classic source→target task: source = "The cat sat on the mat.", target = "Le chat s'est assis sur le tapis." Summarization is the same shape (long source → short target), which is why encoder–decoders suit both.

Sources: Devlin et al. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. NAACL-HLT; Radford et al. (2018). *Improving Language Understanding by Generative Pre-Training*. OpenAI technical report; Raffel et al. (2020). *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. JMLR 21; arXiv:1910.10683

Attention: each token looks at the tokens that matter

THE KEY IDEA INSIDE A TRANSFORMER BLOCK



Self-attention links each token to the relevant others

To represent "*it*", the model must look back to "*animal*". **Self-attention** lets every token pull in information from the others, and the whole sequence is processed at once — so training parallelizes on big machines. It attends to at most a fixed number of tokens at once — its **context window**.

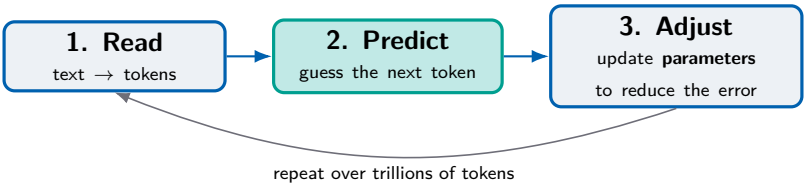
Query, Key, Value — how a token "looks"

Each token forms a **query** (what it is looking for); every token also offers a **key** (what it is about) and a **value** (its content). Attention matches queries to keys, then blends the values — a soft dictionary lookup.
($q = xW_Q,\ k = xW_K,\ v = xW_V$; worked out next.)

Sources: Vaswani et al. (2017). *Attention Is All You Need.* NeurIPS; arXiv:1706.03762

Training: pre-training, then fine-tuning

PIPELINE STEP 3 — HOW THE PARAMETERS ARE LEARNED



Pre-training (self-supervised)

The next token is its own label — no human annotation. This **pre-training** does the heavy lifting: thousands of GPUs, weeks of compute, trillions of tokens.

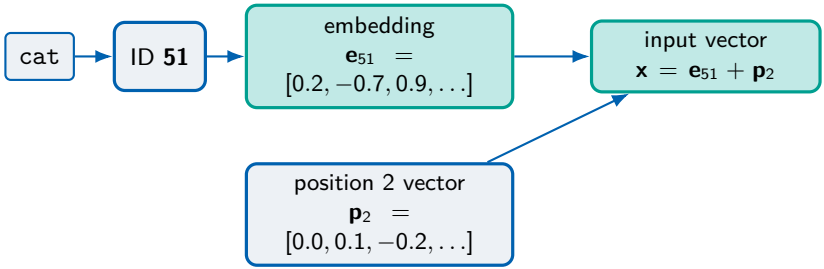
Then fine-tuning / post-training

A shorter stage — instruction tuning and **RLHF** — makes it follow instructions and behave safely. Using the finished model is **inference**.

Sources: Radford et al. (2019). *Language Models are Unsupervised Multitask Learners.* OpenAI technical report; Ouyang et al. (2022). *Training Language Models to Follow Instructions with Human Feedback.* NeurIPS; arXiv:2203.02155

Token IDs, embeddings, and positions

HOW "CAT" BECOMES A VECTOR



Embeddings are learned

The embedding table E has one row per token ID. The rows start random and are **updated during training** — so tokens used in similar ways end up with similar vectors.

Positions are added in

Attention has no built-in sense of order, so a **positional vector** is added: $\mathbf{x}_i = \mathbf{e}_{\text{id}} + \mathbf{p}_i$. Now "cat" at position 2 differs from "cat" elsewhere.

Sources: Bengio et al. (2003). *A Neural Probabilistic Language Model.* JMLR 3:1137–1155; Vaswani et al. (2017). *Attention Is All You Need.* NeurIPS; arXiv:1706.03762

From token vectors to a next-token probability

EVERY SYMBOL IN ONE PLACE — ONE HEAD, CURRENT POSITION t

$$q_t = x_t W_Q, \quad k_j = x_j W_K, \quad v_j = x_j W_V$$

$$s_{t,j} = \frac{q_t \cdot k_j}{\sqrt{d_k}}$$

$$\alpha_{t,j} = \mathrm{softmax}_j\big(s_{t,j}\big) = \frac{e^{s_{t,j}}}{\sum_{j'} e^{s_{t,j'}}}$$

$$z_t = \sum_{j \le t} \alpha_{t,j}\, v_j$$

$$\ell_t = h_t\, W_U$$

$$p\big(x_{t+1} = w \mid x_{\le t}\big) = \mathrm{softmax}(\ell_t)_w$$

Each symbol

x_t: token vector (embedding + position)
W_Q, W_K, W_V: learned projections
q_t, k_j, v_j: query / key / value
d_k: query–key width (keeps softmax stable)
$\alpha_{t,j}$: attention weights, $\sum_j \alpha_{t,j} = 1$
z_t: context-aware vector at position t
h_t: z_t after the block's feed-forward + norm
W_U: output (unembedding) matrix; often tied to the embeddings
ℓ_t: logits — one score per vocabulary token
$p(\cdot)$: the next-token probabilities

All positions at once: $Z = \mathrm{softmax}\Big(QK^\top / \sqrt{d_k}\Big)\, V$. *A causal mask forces* $\alpha_{t,j} = 0$ *for* $j > t$.

Sources: Vaswani et al. (2017). *Attention Is All You Need.* NeurIPS; arXiv:1706.03762

How attention computes a token's new vector

QUERY · KEY → SOFTMAX → WEIGHTED SUM OF VALUES

New vector for **"sat"** over "the cat sat". The query $q = [1, 0]$, keys k_i, and values v_i are *toy numbers chosen for clean arithmetic* — in a real model each is a **learned projection** of the token's embedding ($q = x\,W_Q,\ k_i = x_i W_K,\ v_i = x_i W_V$):

token	key k_i	score $q \cdot k_i$	weight α_i	value v_i
the	$[1, 0]$	1	0.42	$[2, 0]$
cat	$[0, 1]$	0	0.16	$[0, 2]$
sat	$[1, 1]$	1	0.42	$[1, 1]$

$$z_{\text{sat}} = \sum_i \alpha_i v_i \approx [1.26,\ 0.74]$$

z_{sat} = "sat"'s new *context-aware* vector: a blend of the values, weighted by how relevant each token is.

The three steps

1. **score** each token: $q \cdot k_i$
2. **softmax** → weights α_i
3. **mix**: $z = \sum_i \alpha_i v_i$

Whole sequence at once:
$\mathrm{softmax}(QK^\top/\sqrt{d_k})\,V$.

. . . then predict the next token

z_{sat} is the *last* token's vector. It passes through the output layer to a **logit for every vocabulary token**; softmax turns those into the **next-token probabilities**.

Sources: Vaswani et al. (2017). *Attention Is All You Need.* NeurIPS; arXiv:1706.03762

The next-token loss: cross-entropy, term by term

WHAT WE MINIMIZE, AND OVER WHAT

$$\ell_t = -\log p_\theta(x_{t+1} \mid x_{\le t}), \qquad \mathcal{L}(\theta) = \frac{1}{|\mathcal{M}|} \sum_{t \in \mathcal{M}} \ell_t$$

The symbols

- $x_{\le t}$: the tokens seen so far; x_{t+1}: the true next token.
- $p_\theta(\cdot)$: the model's predicted probability of that token.
- θ: **all the model's parameters** (the weights).
- $\mathcal{M}$: the set of positions we *count* (the loss mask).

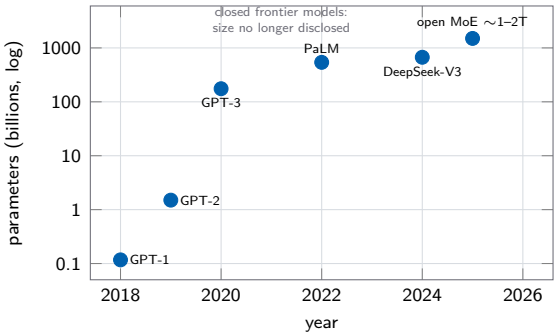
What training adjusts

Minimizing $\mathcal{L}(\theta)$ means changing θ — the parameters of p_θ — by gradient descent, so the model puts *more* probability on the tokens that actually occurred.

Sources: PyTorch (2026). *CrossEntropyLoss*. Official PyTorch documentation; Bengio et al. (2003). *A Neural Probabilistic Language Model*. JMLR 3:1137–1155

Scale: more parameters, more data, more compute

SCALING LAWS, MIXTURE-OF-EXPERTS, EMERGENT ABILITIES



Scaling laws

More **parameters**, data, and compute give predictably better models — and occasionally **emergent abilities** (skills that appear only past a certain scale).

Mixture-of-Experts (MoE)

Big models use a small **learned router** to send each token to its top-1–2 "expert" sub-networks: huge total size, small active part. (Top *closed* models no longer publish their size.)

Sources: Wei et al. (2022). *Emergent Abilities of Large Language Models.* TMLR; arXiv:2206.07682; Hoffmann et al. (2022). *Training Compute-Optimal Large Language Models.* NeurIPS; arXiv:2203.15556; Meta AI (2025). *The Llama 4 Herd.* ai.meta.com/blog/llama-4-multimodal-intelligence

Hallucination: confident, fluent — and sometimes wrong

A LIMIT OF NEXT-TOKEN PREDICTION

Why it happens

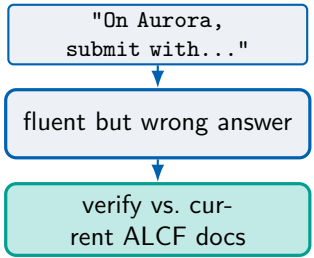
Training rewards output that *sounds* right, not output that *is* right. There is no built-in fact-checker, and training data has a **knowledge cutoff** date.

A real ALCF example

Asked for an Aurora job script, a model may confidently produce `#COBALT`/Slurm directives or outdated modules — but Aurora uses `PBS`. Fluent, and wrong: a **hallucination**.

So always verify

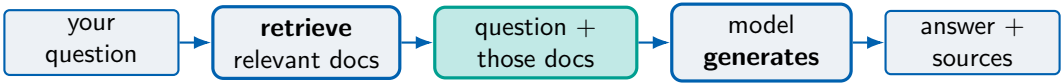
Check facts — especially recent or site-specific details — before you rely on them.



Sources: Farquhar, Kossen, Kuhn & Gal (2024). *Detecting Hallucinations in Large Language Models Using Semantic Entropy*. Nature 630:625–630; doi:10.1038/s41586-024-07421-0; Kalai et al. (2025). *Why Language Models Hallucinate*. arXiv:2509.04664

Retrieval-Augmented Generation (RAG): give it the facts

GROUNDING ANSWERS IN REAL DOCUMENTS



RAG changes what the model can see — not its parameters

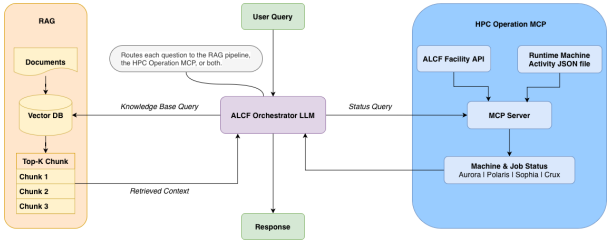
Before answering, the system **retrieves** trustworthy documents and adds them to the prompt (its **context**). Answers become current, grounded, and citable — without retraining.

Sources: Lewis et al. (2020). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.* NeurIPS; arXiv:2005.11401

AskALCF: an AI support assistant for ALCF users

RETRIEVAL + LIVE FACILITY DATA

AskALCF answers ALCF usage questions. An orchestrator LLM routes each question to document **RAG** and/or a live-status tool via the **Model Context Protocol (MCP)** — an open standard (Anthropic, 2024) that lets AI models connect to external tools and data — then answers with sources.



Official AskALCF architecture, ALCF.

Knowledge base it retrieves from

HPC user guides	**ALCF (first priority)**, OLCF, NERSC, LLNL
Schedulers	PBS, Slurm
GPU / compilers	CUDA, SYCL, AMD HIP, Intel oneAPI
Live status	Aurora, Polaris, Sophia, Crux (via MCP)

Try it — Web App

`ask.alcf.anl.gov`

Responses are automated and may not be accurate, complete, or current; confirm critical information with official ALCF docs or support.

Sources: Argonne Leadership Computing Facility (2026). *AskALCF: An AI Support Assistant*. docs.alcf.anl.gov/support/get-help/askalcf; Anthropic (2024). *Introducing the Model Context Protocol*. anthropic.com/news/model-context-protocol

The field moves fast

A FEW TRENDS WORTH KNOWING

Open-weight models rival closed ones: DeepSeek and Qwen trail the best closed models by months, at far lower cost.

Reasoning is now a built-in "thinking" mode: flagships expose an adjustable reasoning effort (GPT-5, Gemini 3, Claude, DeepSeek).

Context windows keep growing: ∼1M tokens is now the norm; Llama 4 Scout reaches 10M — whole codebases at once.

Agents with tools, knowledge databases, and skills: models that plan and act in a loop — calling **tools** (via MCP), retrieving from **knowledge databases** (RAG), and loading packaged **skills** on demand. AskALCF is an example.

Sources: DeepSeek-AI (2025). *DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning*. Nature 645:633–638; arXiv:2501.12948; Meta AI (2025). *The Llama 4 Herd*. ai.meta.com/blog/llama-4-multimodal-intelligence; Gemini Team, Google (2024). *Gemini 1.5: Unlocking Multimodal Understanding Across Millions of Tokens of Context*. arXiv:2403.05530; Anthropic (2024). *Introducing the Model Context Protocol*. anthropic.com/news/model-context-protocol; Anthropic (2025). *Agent Skills*. docs.claude.com/en/docs/agents-and-tools/agent-skills

Hands-on micro-lab: tokenization and training

A GUIDED 6–8 MINUTE NOTEBOOK

Part A — Tokenization

- split text into **tokens** and inspect the **token IDs**
- see how boundaries and normalization change the tokens

Part B — Training loop

Train a tiny decoder (10,688 parameters) with **next-token prediction**:

shift → predict → loss → update → generate

Get the code

Repo (open source)

`github.com/argonne-lcf/ATPESC_MachineLearning`

Notebook

`lab/atpesc_llm_micro_lab.ipynb`

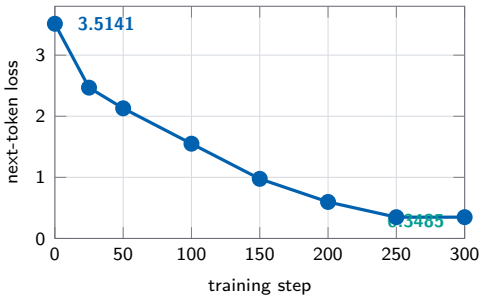
Also try: ask.alcf.anl.gov

A deterministic, CPU-only teaching model — it shows the mechanism, not frontier-scale capability.

Sources: Argonne LCF (2026). *ATPESC MachineLearning: 03_languagemodels (MiniLLM).* github.com/argonne-lcf/ATPESC_MachineLearning; Local micro-lab artifact (deterministic, CPU-only, seed 2026)

What you'll see: the loss drops as it learns

MICRO-LAB OBSERVED RESULT



Seeded sample after training

```
aurora trains models.
aurora trains weighs context.
exp
```

Model under the microscope

10,688 parameters; one decoder block; character vocabulary; 28 local lines repeated deterministically.

Same training loop as a real LLM — radically different data, model, and scale.

Sources: Radford et al. (2019). *Language Models are Unsupervised Multitask Learners.* OpenAI technical report; Local micro-lab artifact (deterministic, CPU-only, seed 2026)

Key terms at a glance

THE VOCABULARY OF THIS TALK

- **Token / tokenization** — text split into word-pieces the model reads.
- **Embedding** — the learned vector a token maps to.
- **Transformer / attention** — the architecture; attention links relevant tokens.
- **Context window** — how many tokens it sees at once.
- **Language modeling / next-token prediction** — the core task.
- **Inference** — using the trained model to generate.
- **Pre-training** — self-supervised learning on huge text.
- **Fine-tuning / post-training (RLHF)** — makes it follow instructions.
- **Parameters / weights** — the learned numbers; grow via **scaling laws**; **MoE** keeps the active part small.
- **Emergent abilities** — skills that appear with scale.
- **Hallucination** — confident but wrong output.
- **RAG / MCP / skills** — grounding, tool protocol, packaged know-how.

Key takeaways

FIVE THINGS TO REMEMBER

- One idea — **next-token prediction** — at massive scale produces broad capability.
- The pipeline is **tokenization** → **Transformer/attention** → **training** (pre-training + fine-tuning) → **inference**.
- It learned *patterns of language*, not a database of facts — so it can be confidently wrong (**hallucination**).
- Ground it with **RAG / MCP / skills** and **verify** before you trust it.
- Best way to learn: try the hands-on notebook and AskALCF today.

Sources: Radford et al. (2019). *Language Models are Unsupervised Multitask Learners.* OpenAI technical report; Lewis et al. (2020). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.* NeurIPS; arXiv:2005.11401

Acknowledgements

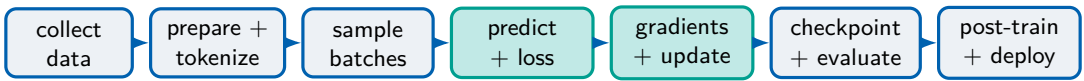
ARGONNE TRAINING PROGRAM ON EXTREME-SCALE COMPUTING

Produced by Argonne National Laboratory, a U.S. Department of Energy Laboratory managed by UChicago Argonne, LLC under contract DE-AC02-06CH11357.

Special thanks to the National Energy Research Scientific Computing Center (NERSC) and Oak Ridge Leadership Computing Facility (OLCF) for the use of their resources during the training event.

Backup: the full training pipeline

PIPELINE OVERVIEW



The same objective sits inside a larger engineered process

Data and preprocessing define examples. Prediction and optimization update weights. Evaluation measures behavior with weights held fixed.

Sources: Narayanan et al. (2021). *Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM.* SC'21; arXiv:2104.04473; Duan et al. (2024). *Efficient Training of Large Language Models on Distributed Infrastructures: A Survey.* arXiv:2407.20018

Backup: two ways to split text into tokens

WORD-LEVEL VS. SUBWORD

Method	"The cat sat on the mat."	rare word "unbelievable"
Word-level	The | cat | sat | on | the | mat | . | [UNK]	(not in vocab)
Subword (BPE)	The | cat | sat | ...	un | believ | able

Why modern LLMs use subword tokenization

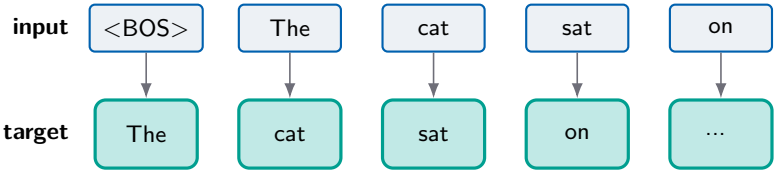
Word-level vocabularies explode and still miss new words (`[UNK]`). **Subword** methods (BPE, WordPiece, Unigram) keep common words whole but break rare ones into known pieces — a fixed vocabulary that can spell *any* word.

Character-level (one token per character) also avoids unknowns, but makes sequences very long.

Sources: Sennrich et al. (2016). *Neural Machine Translation of Rare Words with Subword Units.* ACL; arXiv:1508.07909; Kudo & Richardson (2018). *SentencePiece: A Simple and Language Independent Subword Tokenizer and Detokenizer for Neural Text Processing.* EMNLP; arXiv:1808.06226

Backup: shifted targets and the causal mask

ONE PASS, MANY NEXT-TOKEN PREDICTIONS



Each target uses *all* tokens to its left

The arrows show position alignment, but predicting "sat" uses the whole left context (<BOS>, The, cat) — not just the token directly above.

Causal mask

A mask inside attention that blocks each position from seeing *future* tokens. So one forward pass yields a valid next-token prediction at every position, without peeking ahead.

Sources: Vaswani et al. (2017). *Attention Is All You Need.* NeurIPS; arXiv:1706.03762; Radford et al. (2019). *Language Models are Unsupervised Multitask Learners.* OpenAI technical report

Backup: distributed training — one update across many devices

THE TRAINING STEP, THEN HOW IT IS PARTITIONED

```
logits = model(inputs)
loss = cross_entropy(logits, targets)

optimizer.zero_grad() # clear old gradients
loss.backward()       # backprop: compute
                      #   gradients of loss
optimizer.step()      # update parameters
                      #   using the gradients
```

Ways to split the work

- **Data parallel** — split the *batch*; every device has a full model copy; average gradients (all-reduce).
- **Tensor parallel** — split a layer's weight *matrices* across devices.
- **Pipeline parallel** — put different *layers* on different devices.
- **Sharding (ZeRO/FSDP)** — split optimizer state, gradients, or parameters to save memory.

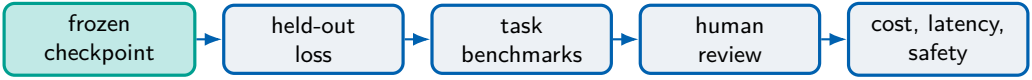
Sources: Narayanan et al. (2021). *Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM.* SC'21; arXiv:2104.04473; Duan et al. (2024). *Efficient Training of Large Language Models on Distributed Infrastructures: A Survey.* arXiv:2407.20018

Backup: checkpoints and evaluation

WHAT A "CHECKPOINT" IS, AND WHY WE EVALUATE IT

A checkpoint is a saved snapshot of the parameters

During long training we periodically save θ to disk. A **checkpoint** lets us (1) resume after a crash, (2) evaluate the model *with weights frozen*, and (3) deploy a chosen snapshot.



Why evaluate beyond training loss

Low training loss only means "fits the training text." Benchmarks, human review, and system metrics test what we actually care about — interpret each with its dataset, metric, and setup (e.g. MMLU, HumanEval).

Sources: Bommasani et al. (2021). *On the Opportunities and Risks of Foundation Models.* arXiv:2108.07258

Backup: pre-training vs. supervised fine-tuning (SFT)

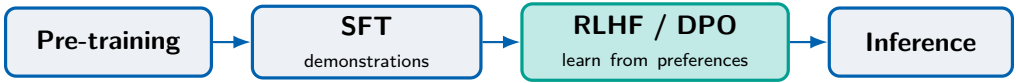
SAME LOSS, DIFFERENT DATA AND TARGET

Pre-training

- **Data:** massive *raw* text (self-supervised).
- **Loss:** next-token on *every* token.
- **Learns:** language and world knowledge.

Supervised fine-tuning (SFT)

- **Data:** curated *(prompt → ideal response)* pairs.
- **Loss:** the *same* next-token loss, but counted only on the **response** tokens.
- **Learns:** to follow instructions.

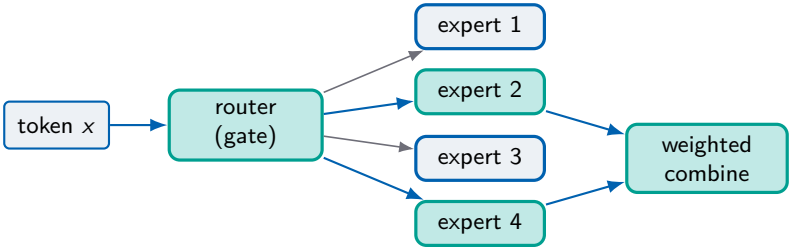


Then preference methods (RLHF, DPO) align behavior; Tülu 3 is one current open recipe.

Sources: Ouyang et al. (2022). *Training Language Models to Follow Instructions with Human Feedback.* NeurIPS; arXiv:2203.02155; Rafailov et al. (2023). *Direct Preference Optimization: Your Language Model Is Secretly a Reward Model.* NeurIPS; arXiv:2305.18290; Lambert et al. (2025). *Tülu 3: Pushing Frontiers in Open Language Model Post-Training.* arXiv:2411.15124

Backup: Mixture-of-Experts (MoE) routing

A SMALL LEARNED ROUTER SENDS EACH TOKEN TO A FEW EXPERTS



How it routes (per token, per layer)

Score each expert with a tiny linear router, $g = \mathrm{softmax}(x \cdot W_r)$; keep the **top-$k$** (here 2 of 4); run only those; combine their outputs weighted by g.

Learned, sparse, balanced

W_r is **learned** with the model. Only k of N experts run per token $\Rightarrow$ huge total size, small active compute. A **load-balancing** loss keeps experts evenly used.

Sources: Fedus, Zoph & Shazeer (2022). *Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity.* JMLR; arXiv:2101.03961