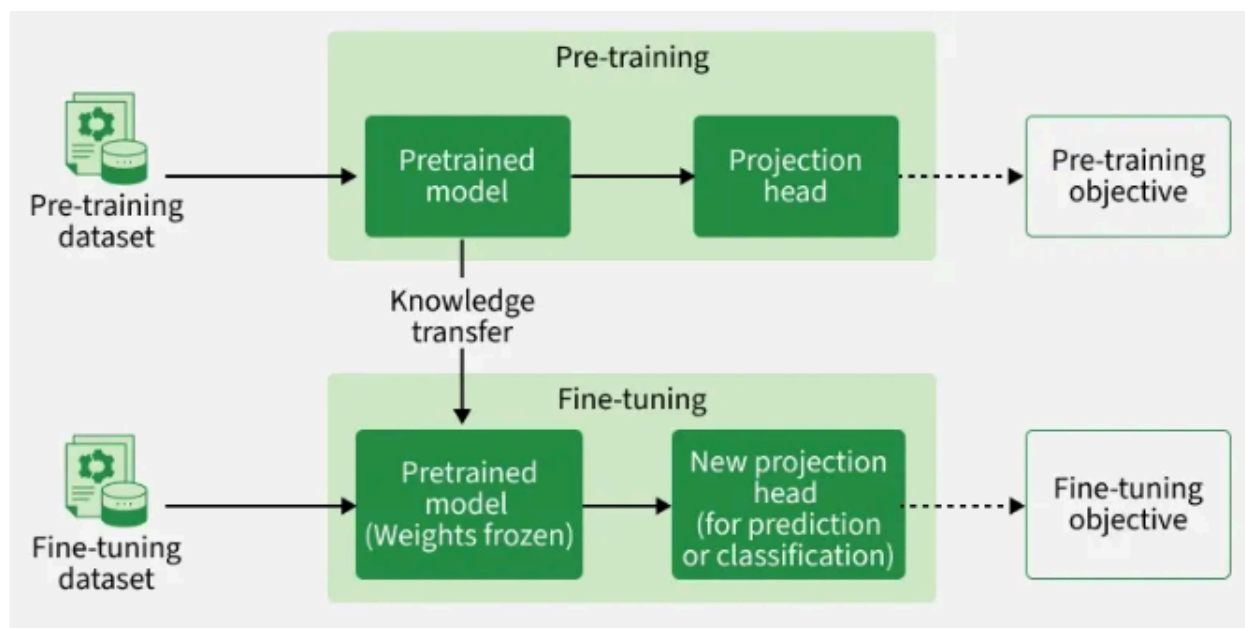


LLM Training-এর ২টি প্রধান ধাপ: Pre-training বনাম Fine-tuning

একটি আধুনিক LLM বানাতে সাধারণত দুইটা স্তর থাকে:

1. **Pre-training** → মডেলকে “ভাষার সাধারণ জ্ঞান” শেখানো (foundation তৈরি)
2. **Fine-tuning** → নির্দিষ্ট কাজ/ডোমেইনের জন্য মডেলকে “বিশেষজ্ঞ” করা



১) Pre-training (ভিত্তি তৈরি)

Pre-training আসলে কী?

Pre-training হলো মডেলকে বিশাল, বৈচিত্র্যময় টেক্সট ডেটা দিয়ে এমনভাবে ট্রেন করা, যাতে সে ভাষার সাধারণ নিয়ম, ধারণা, এবং প্যাটার্ন শিখে ফেলে। এই ধাপে মডেল সাধারণত অগণিত বাক্য/ডকুমেন্ট পড়ে।

ডেটা কোথা থেকে আসে?

Pre-training data সাধারণত আসে ওপেন ওয়েব/কর্পাস থেকে—যেমন:

- **Common Crawl** (ওপেন ওয়েব ক্রল ডেটা; অনেক LLM প্রি-ট্রেনিংয়ে ব্যবহৃত) (commoncrawl.org)
- **Wikipedia**, বই, ওয়েব আর্টিকেল ইত্যাদি (বিভিন্ন মিশ্র)
- অনেক ক্ষেত্রে “ওয়েব-স্টাইল টেক্সট” ধরনের কর্পাসও থাকে (প্ল্যাটফর্ম-সোর্সড টেক্সটের মিশ্রণ)

Dataset	Quantity (tokens)	Weight in training mix	Epochs elapsed when training for 300B tokens
Common Crawl (filtered)	410 billion	60%	0.44
WebText2	19 billion	22%	2.9
Books1	12 billion	8%	1.9
Books2	55 billion	8%	0.43
Wikipedia	3 billion	3%	3.4

Pre-training এ কী শেখানো হয়?

সবচেয়ে কমন লক্ষ্য হলো **Next-token prediction** (তোমার ভাষায় “word completion”)—মডেলকে শেখানো হয়:

“এ পর্যন্ত যে টোকেনগুলো দেখেছি, তার ভিত্তিতে পরের টোকেনটি কী হওয়া উচিত?”

উদাহরণ:

“The lion is in the ...” → সম্ভাব্য “forest / zoo / wild”

মডেল আসলে একটা সম্ভাব্যতার বন্টন (probability distribution) শেখে—একটা শব্দ মুখস্থ করে না।

Input tokens → Model → Probability over next token
→ Pick next token

তাহলে শুধু **next-word** শিখে কীভাবে এত কাজ পারে?

এটা LLM-এর গুরুত্বপূর্ণ ধারণা: **emergent capabilities / generalization**

যেহেতু মডেল বিপুল টেক্সট পড়ে ভাষার প্যাটার্ন, facts, নির্দেশনা-ধাঁচ, প্রশ্নোত্তর ধাঁচ ইত্যাদি দেখে—ফলে “শুধু next-token” objective থেকেও অনেক ক্ষমতা তৈরি হয় (যেমন সারসংক্ষেপ, অনুবাদ, QA), বিশেষ করে স্কেল (data + compute + parameters) বাড়লে। Common Crawl—ভিত্তিক ডেটা LLM pretraining-এর বড় ভিত্তি হিসেবে ব্যাপকভাবে ব্যবহৃত—এটাই এই generalization ঘটানোর পরিবেশ তৈরি করে। (facctconference.org)

Pre-training কেন এত ব্যয়বহুল?

কারণ এখানে দরকার:

- অত্যন্ত বড় ডেটা
- বড় মডেল (বহু parameter)
- বহু GPU/TPU compute
- সপ্তাহ/মাসব্যাপী ট্রেনিং রান

এটার খরচ নিয়ে বিভিন্ন জায়গায় **estimate** থাকে। উদাহরণ হিসেবে GPT-3 ট্রেনিং compute cost নিয়ে নানা অনুমান পাওয়া যায়—কিছু বিশ্লেষণে ~\$0.5M থেকে ~\$4.6M পর্যন্ত রেঞ্জ বলা হয়েছে (hardware/efficiency assumption ভেদে)। ([Andreessen Horowitz](https://arxiv.org/abs/2204.05914))

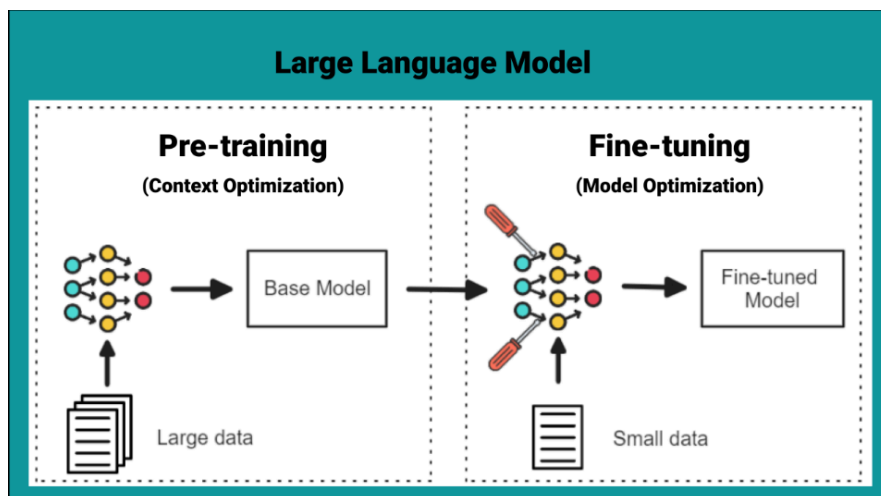
এখানে গুরুত্বপূর্ণ ব্যাপার: এগুলো **estimate**, exact অফিসিয়াল “মোট” খরচ একেকভাবে গণনা করা হয় (compute-only বনাম total engineering cost)।

Pre-training শেষ হলে কী পাওয়া যায়?

এ ধাপ শেষে তুমি পাবে একটি:

- **Base Model / Foundation Model**

যার ভাষা-জ্ঞান সাধারণ, কিন্তু আচরণ “generic”—মানে এটা সবকিছু কিছুটা পারে, কিন্তু নির্দিষ্ট কাজ/ডোমেইনে সেরা নাও হতে পারে।



২) Fine-tuning (বিশেষজ্ঞ করা)

Fine-tuning কেন লাগে?

Pre-trained foundation model “সাধারণ জ্ঞান” রাখলেও—

- তোমার কোম্পানির **internal policy**, **product catalog**, **SOP**, গ্রাহক **FAQ**, আইনি স্টাইল—এসব জানে না
- নির্দিষ্ট টোন/ফরম্যাট/রুল মেনে উত্তর দেওয়া গ্যারান্টি নয়
- নির্দিষ্ট কাজ (classification, support, legal drafting) এ consistent হতে পারে না

তাই তুমি model-কে নিজের লক্ষ্য অনুযায়ী **adjust** করো—এটাই fine-tuning।

Fine-tuning ডেটা কেমন হয়?

Pre-training এ ডেটা অনেক সময় “unsupervised / unlabeled” (কেবল টেক্সট)।

Fine-tuning এ সাধারণত **labeled** বা **structured** ডেটা থাকে—যেমন:

- (Instruction, Response) pair
- (Input, Label) pair
- QA dataset, support ticket → correct resolution
- Legal clause → rewritten clause + rationale

Fine-tuning এর ২টি গুরুত্বপূর্ণ ধরন

A) Instruction Fine-tuning (IFT)

এখানে মডেলকে শেখানো হয় নির্দেশনা মানা—যেমন:

- “এই প্যারাগ্রাফটা ৩ লাইনে summarize করো”
- “English থেকে French অনুবাদ করো”
- “এই প্রশ্নের উত্তর ধাপে ধাপে দাও”

ডেটা ফরম্যাট:

Instruction + Input → Ideal Output

B) Classification Fine-tuning

এখানে মডেলকে শেখানো হয় ক্যাটাগরি/লেবেল দিতে—যেমন:

- Spam vs Not spam
- Positive vs Negative sentiment
- Topic: Sports/Politics/Tech

ডেটা ফরম্যাট:

Text → Label