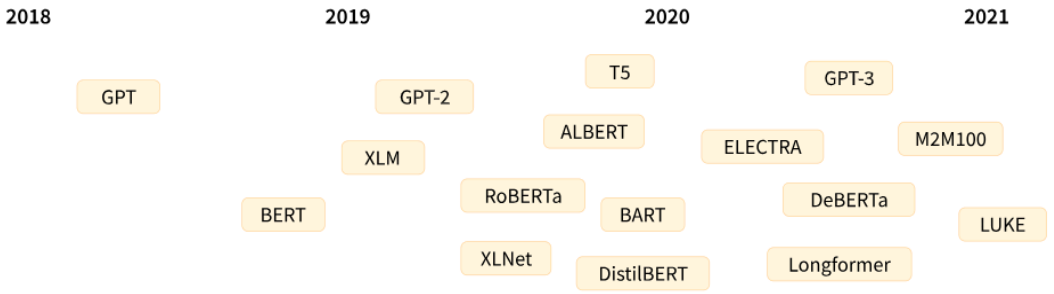
Transformers কীভাবে কাজ করে?

এই সেকশনে আমরা Transformer মডেলের আর্কিটেকচার (architecture) নিয়ে ধারণা নেব এবং বিশেষ করে **attention**, **encoder-decoder architecture**, এবং এগুলোর সাথে সম্পর্কিত আরও কিছু গুরুত্বপূর্ণ কনসেপ্ট বুঝব।

নোট: এই অংশটা তুলনামূলকভাবে টেকনিক্যাল। প্রথমবারে সবকিছু পুরোপুরি ক্লিয়ার না হলেও সমস্যা নেই—কোর্সের পরে অংশগুলোতে আবার এগুলোতে ফিরেও আসা হবে ।



১) Transformer-এর সংক্ষিপ্ত ইতিহাস

Transformer আর্কিটেকচার প্রথম পরিচিত হয় জুন ২০১৭-এ। মূল গবেষণার ফোকাস ছিল **translation** (এক ভাষা থেকে আরেক ভাষায় অনুবাদ) টাস্কের ওপর। এরপর কয়েকটি গুরুত্বপূর্ণ মডেল আসে, যেগুলো Transformer-কে জনপ্রিয় করে তোলে:

- জুন ২০১৮**: GPT**
  প্রথম pretrained Transformer মডেলগুলোর একটি। বিভিন্ন NLP টাস্কে fine-tune করে শক্তিশালী ফলাফল দেখায়।
- অক্টোবর ২০১৮**: BERT**
  বড় pretrained মডেল, বাক্য/টেক্সটের representation আরও ভালোভাবে ধরতে ডিজাইন করা। (এটা পরের চ্যাপ্টারে আরও পরিষ্কার হবে।)
- ফেব্রুয়ারি ২০১৯**: GPT-2**
  GPT-এর বড় ও উন্নত ভার্সন। নৈতিক/এথিক্যাল উদ্বেগের কারণে শুরুতে পুরোটা তৎক্ষণাৎ প্রকাশ করা হয়নি।
- অক্টোবর ২০১৯**: T5**
  sequence-to-sequence Transformer আর্কিটেকচারকে multi-task স্টাইলে ব্যবহার করার জনপ্রিয় প্রয়োগ।

- মে ২০২০**: GPT-3**
  GPT-2-এর আরও বড় সংস্করণ। অনেক টাস্কে fine-tune ছাড়াই ভালো পারফর্ম করতে পারে—যাকে সাধারণভাবে zero-shot ক্ষমতা বলা হয়।
- জানুয়ারি ২০২২**: InstructGPT**
  GPT-3-এর এমন একটি ভার্সন যেটাকে "ইনস্ট্রাকশন ঠিকমতো ফলো করা" শেখাতে অতিরিক্তভাবে ট্রেন করা হয়েছে।
- জানুয়ারি ২০২৩**: Llama**
  বহু ভাষায় টেক্সট জেনারেট করতে সক্ষম বড় ভাষা মডেল।
- মার্চ ২০২৩**: Mistral**
  ৭ বিলিয়ন প্যারামিটার মডেল; বিভিন্ন বেঞ্চমার্কে শক্তিশালী পারফরম্যান্সের জন্য পরিচিত। দ্রুত inference এর জন্য grouped-query attention এবং দীর্ঘ সিকোয়েন্স হ্যান্ডেল করতে sliding window attention-এর মতো ধারণা ব্যবহারের কথা উল্লেখ করা হয়।
- মে ২০২৪**: Gemma 2**
  ২B থেকে ২৭B প্যারামিটারের lightweight ওপেন মডেল পরিবার। interleaved local-global attention এবং group-query attention-এর মতো ধারণা ব্যবহারের কথা বলা হয়েছে; ছোট মডেলগুলোতে knowledge distillation ব্যবহার করে তুলনামূলক বড় মডেলের কাছাকাছি পারফরম্যান্স আনার চেষ্টা করা হয়।
- নভেম্বর ২০২৪**: SmolLM2**
  ছোট আকারের (১৩৫M থেকে ১.৭B প্যারামিটার) হলেও ভালো পারফরম্যান্সের জন্য পরিচিত; বিশেষ করে mobile/edge device-এর মতো কম রিসোর্স পরিবেশে সম্ভাবনা তৈরি করে।

এই তালিকাটা "সম্পূর্ণ ইতিহাস" নয়—কেবল বিভিন্ন ধরনের Transformer মডেলের কয়েকটা উদাহরণ, যাতে আপনি বড় ধারাটা বুঝতে পারেন।

২) **Transformer** মডেলের প্রধান তিনটি পরিবার

Transformer মডেলগুলোকে সাধারণভাবে ৩ ভাগে গ্রুপ করা যায়:

1. **GPT-like (Auto-regressive Transformers)**
   বাক্যের পরের টোকেন/শব্দ ভবিষ্যদ্বাণী করে ধাপে ধাপে টেক্সট তৈরি করে। Text generation-এ খুব শক্তিশালী।
2. **BERT-like (Auto-encoding Transformers)**
   বাক্যের ভেতরের অর্থ/কনটেক্সট বুঝতে বেশি ফোকাস করে। Classification, NER, semantic understanding টাইপ কাজে খুব কার্যকর।
3. **T5-like (Sequence-to-sequence Transformers)**
   ইনপুট সিকোয়েন্স নিয়ে আউটপুট সিকোয়েন্স বানায়। Translation, summarization, text-to-text টাস্কে খুব সুবিধাজনক।

কোর্সের পরের অংশে এই তিন পরিবারকে আরও গভীরভাবে দেখা হবে।

৩) **Transformers** কেন **“Language Model”** হিসেবে ট্রেন হয়?

উপরে উল্লেখ করা GPT, BERT, T5—সবগুলোই মূলত **language model** হিসেবে **pretrained**। মানে:

- প্রচুর পরিমাণ raw text ডেটা দিয়ে
- self-supervised পদ্ধতিতে
- ভাষার pattern/structure/পরিসংখ্যানগত নিয়ম শিখে

Self-supervised learning কী?

এটা এমন ট্রেনিং যেখানে “লেবেল” মানুষ আলাদা করে দেয় না। বরং ইনপুট থেকেই টার্গেট/উদ্দেশ্য তৈরি হয়। ফলে বিশাল ডেটায় ট্রেনিং করা সম্ভব হয়, কারণ মানুষের হাতে লেবেলিং লাগে না।

কিন্তু সমস্যা হলো: pretrained ভাষা-বোঝা মডেল সবসময় সরাসরি নির্দিষ্ট বাস্তব কাজের জন্য প্রস্তুত থাকে না। তাই পরে আমরা করি:

Transfer Learning / Fine-tuning

এখানে:

- সাধারণ pretrained মডেলকে
- নির্দিষ্ট টাস্কের labelled ডেটা দিয়ে
- supervised ভাবে fine-tune করা হয়
  যেমন sentiment analysis, spam detection, QA ইত্যাদি।

৪) দুটি ক্লাসিক **Training Objective: Causal LM** বনাম **Masked LM**

Transformer-based language model ট্রেন করার দুইটা বহুল ব্যবহৃত উদ্দেশ্য হলো:

ক) **Causal Language Modeling (Next-token prediction)**

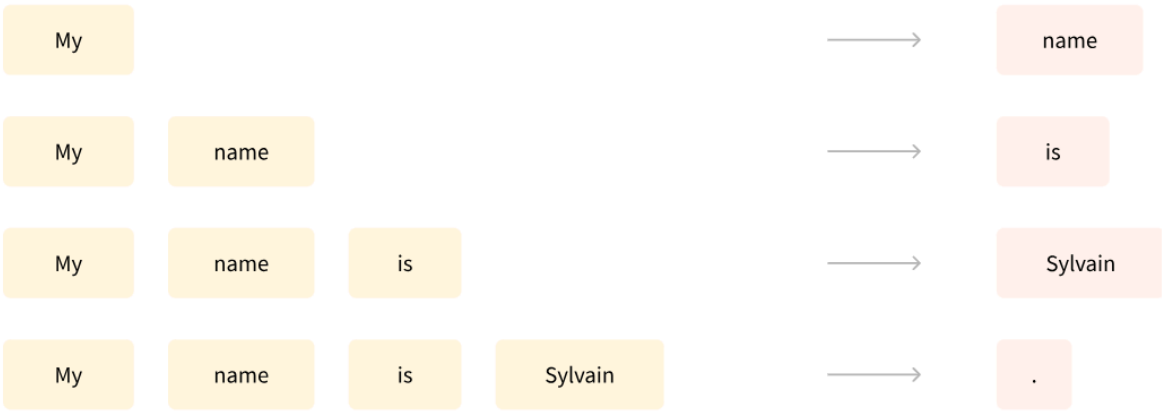
এখানে লক্ষ্য:

- আগের n শব্দ/টোকেন দেখে পরের শব্দ/টোকেন প্রেডিক্ট করা

এটাকে “causal” বলা হয় কারণ:

- আউটপুট নির্ভর করে অতীত ও বর্তমান ইনপুটের উপর
- ভবিষ্যৎ টোকেন দেখা নিষিদ্ধ (না হলে cheating হয়ে যাবে)

এই স্টাইল GPT-like মডেলের সাথে বেশি যায় এবং text generation-এর ভিত্তি।

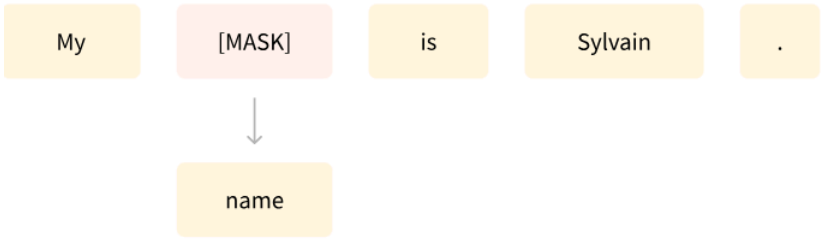


খ) Masked Language Modeling (Fill-in-the-blank)

এখানে লক্ষ্য:

- বাক্যের কিছু শব্দ mask করে দেওয়া হয়
- মডেল সেই mask করা শব্দ প্রেডিক্ট করে

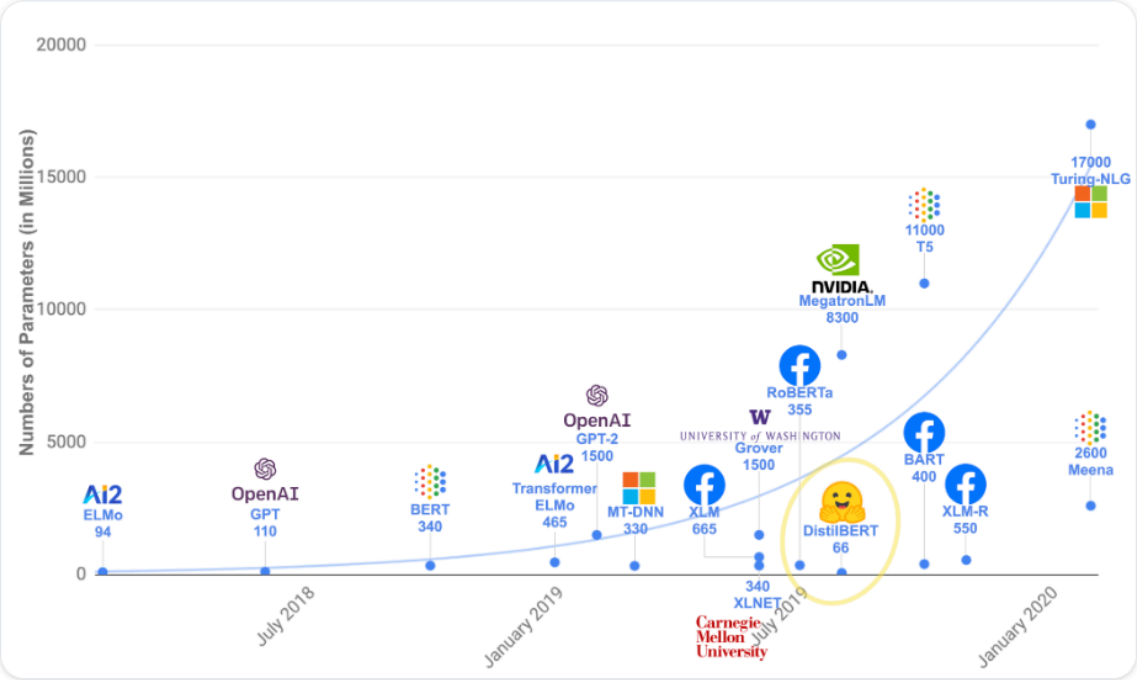
এই স্টাইল BERT-like মডেলের সাথে বেশি যায় এবং context understanding/representation শেখায়।



৫) Transformers সাধারণত “বড়” মডেল

Transformer মডেলে পারফরম্যান্স বাড়ানোর একটা সাধারণ কৌশল হলো:

- মডেলের size (parameters) বাড়ানো
- pretrained ডেটার পরিমাণ বাড়ানো



কিছু ব্যতিক্রম আছে (যেমন DistilBERT-এর মতো distilled/compact মডেল), কিন্তু ইন্ডাস্ট্রির বড় অংশে স্কেল বাড়ানোই প্রধান পথ হিসেবে দেখা গেছে।

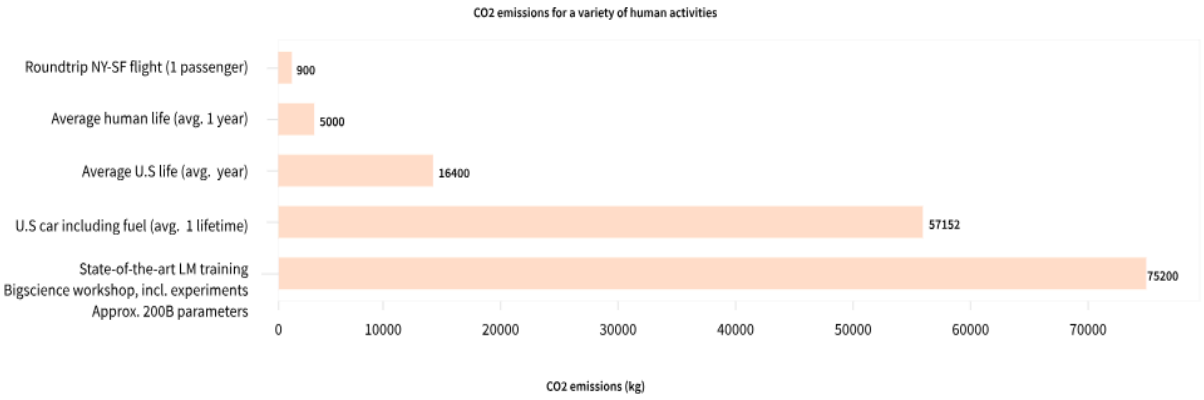
৬) বড় মডেলের খরচ: সময়, কম্পিউট, পরিবেশগত প্রভাব

বড় ভাষা মডেল ট্রেনিং করতে লাগে:

- অনেক ডেটা
- অনেক GPU/TPU সময়
- অনেক বিদ্যুৎ ও খরচ

এর ফল হিসেবে **carbon footprint**-ও বাড়ে। শুধু একবার ট্রেনিং নয়—সেরা hyperparameter পেতে বহু ট্রায়াল চালালে footprint আরও বেড়ে যেতে পারে।

এখন একটা গুরুত্বপূর্ণ যুক্তি আসে:



যদি সবাই বারবার “শূন্য থেকে” ট্রেন করে?

ধরুন:

- প্রতিবার গবেষক দল/স্টুডেন্ট/কোম্পানি নতুন করে scratch থেকে ট্রেন করে
  তাহলে বিশ্বজুড়ে অপ্রয়োজনীয়ভাবে বিশাল compute খরচ হবে এবং পরিবেশগত প্রভাবও বাড়বে।

৭) তাই **pretrained** মডেল শেয়ার করা এত গুরুত্বপূর্ণ

এই কারণেই pretrained language model শেয়ার করা অত্যন্ত গুরুত্বপূর্ণ:

- একবার ট্রেন করা weights শেয়ার করা হলে
- অন্যরা সেই weights-এর উপর fine-tune করে কাজ করতে পারে
- মোট compute খরচ কমে
- কমিউনিটির মোট carbon footprint কমে

৮) **Training carbon footprint** মাপার টুল

মডেল ট্রেনিং-এর কার্বন ইমপ্যাক্ট অনুমান/ট্র্যাক করার জন্য কিছু টুল ব্যবহার করা যায়, যেমন:

- ML CO2 Impact
- Code Carbon (এবং Transformers-এ ইন্টিগ্রেশনও থাকে)

এসব দিয়ে আপনি ট্রেনিং চলাকালে একটি emissions রিপোর্ট (যেমন emissions.csv) টাইপ আউটপুট পেতে পারেন—যেটা ট্রেনিংয়ের আনুমানিক footprint বুঝতে সাহায্য করে।

Transfer Learning (ট্রান্সফার লার্নিং)

Pretraining কী?

Pretraining মানে হলো কোনো মডেলকে “শূন্য থেকে” (from scratch) ট্রেন করা। এখানে:

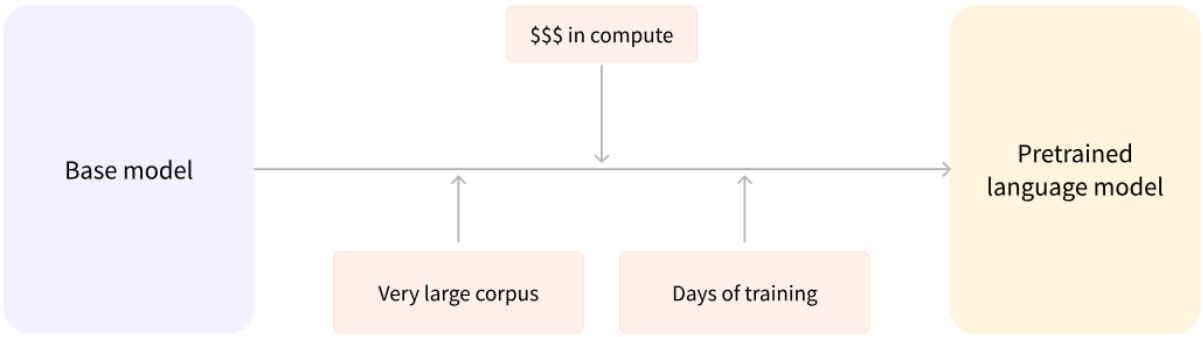
- মডেলের **weights** শুরুতে সম্পূর্ণ র‍্যান্ডম থাকে
- মডেল কোনো আগের জ্ঞান ছাড়াই শেখা শুরু করে

ভাষা মডেলের (language model) ক্ষেত্রে pretraining সাধারণত করা হয় বিশাল পরিমাণ raw text ডেটার উপর—যেমন বই, ওয়েব পেজ, আর্টিকেল, কোড ইত্যাদি।

কেন **Pretraining** এত ব্যয়বহুল?

কারণ:

- ডেটা খুব বড় লাগে (একটা বিশাল corpus দরকার)
- ট্রেনিং অনেক সময় নেয় (কখনো সপ্তাহখানেক পর্যন্ত লাগতে পারে)
- GPU/TPU compute দরকার হয় প্রচুর
  ফলে সময় ও টাকা—দুই দিক থেকেই pretraining খুব expensive।



Fine-tuning কী?

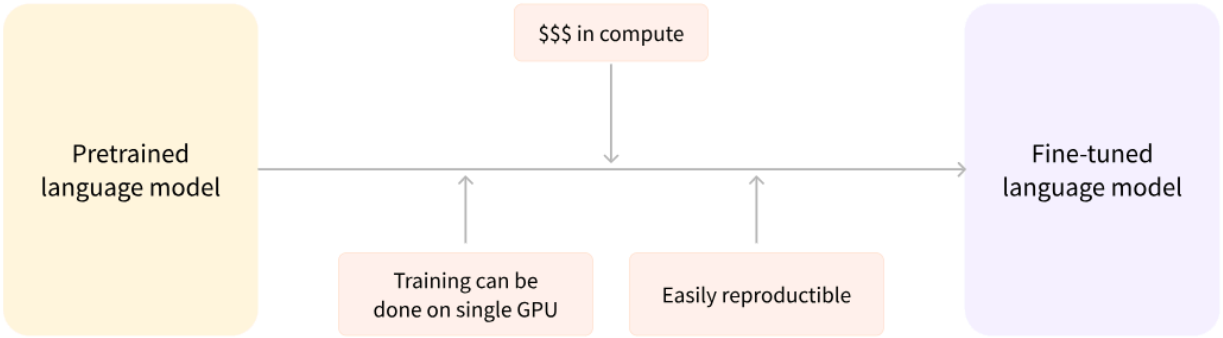
Fine-tuning হলো pretraining হয়ে যাওয়ার পরের ধাপ।

এখানে আপনি:

1. আগে থেকে pretrained একটি language model সংগ্রহ করেন
2. তারপর আপনার নির্দিষ্ট কাজের ডেটাসেট (task-specific dataset) দিয়ে আরও কিছু ট্রেনিং করান

এখন প্রশ্ন আসে—
“শেষ কাজের জন্য শুরু থেকেই scratch থেকে ট্রেন করলেই তো হয়, fine-tuning কেন?”
এর কয়েকটা বড় কারণ আছে।

Fine-tuning কেন লাভজনক?



কারণ ১: Pretrained মডেলের আগেই অনেক জ্ঞান থাকে

Pretrained মডেল আগে থেকেই এমন ডেটায় ট্রেন করা থাকে, যা আপনার কাজের ডেটার সাথে কিছুটা হলেও সম্পর্কযুক্ত/সাদৃশ্যপূর্ণ।

NLP-এর ক্ষেত্রে pretrained মডেল সাধারণত ভাষা সম্পর্কে একটা “স্ট্যাটিস্টিক্যাল বোঝাপড়া” শিখে ফেলে:

- শব্দের ব্যবহার
- বাক্যের গঠন
- কোন শব্দের পরে কোনটা আসতে পারে
- context অনুযায়ী অর্থ বদলানো

ফলে fine-tuning করার সময় মডেল আগের শিখে রাখা জ্ঞান ব্যবহার করে দ্রুত অ্যাডাপ্ট করতে পারে।

কারণ ২: কম ডেটা দিয়েই ভালো ফল পাওয়া যায়

Scratch থেকে ট্রেন করলে বিশাল ডেটা দরকার।
কিন্তু fine-tuning-এ pretrained মডেলের ফাউন্ডেশন প্রস্তুত থাকে, তাই অল্প ডেটা দিয়েই **decent result** পাওয়া যায়।

কারণ ৩: কম সময়, কম রিসোর্স লাগে

Pretraining-এর তুলনায় fine-tuning:

- দ্রুত শেষ হয়
- কম GPU লাগে

- খরচ কম
- সেটআপ/এক্সপেরিমেন্ট করা সহজ

কারণ ৪: “জ্ঞান ট্রান্সফার” হয় বলেই নাম **Transfer Learning**

Pretrained মডেল যে জ্ঞান শিখেছে, সেটা fine-tuning-এর মাধ্যমে আপনার কাজের দিকে স্থানান্তর **(transfer)** হয়। এ জন্যই একে বলা হয় **Transfer Learning**।

উদাহরণ

ধরুন আপনার কাছে:

- ইংরেজি ভাষায় pretrained একটি মডেল আছে
  আপনি এটাকে আবার:
- arXiv (গবেষণাপত্র) কর্পাসে fine-tune করলেন

তাহলে আপনি পাবেন:

- বিজ্ঞান/রিসার্চ ডোমেইন ফোকাসড একটি মডেল

এখানে আপনাকে scratch থেকে বিজ্ঞান-ভাষা শেখাতে হয়নি—মডেলের ভাষাগত ভিত্তি আগে থেকেই ছিল, আপনি শুধু ডোমেইন অনুযায়ী টিউন করেছেন।

Fine-tuning কেন তুলনামূলকভাবে “চিপার”?

Fine-tuning সাধারণত pretraining-এর চেয়ে কম লাগে:

- সময় (time cost)
- ডেটা (data cost)
- টাকা (financial cost)
- পরিবেশগত প্রভাব (environmental cost)

এবং সবচেয়ে বড় সুবিধা:

- বিভিন্ন fine-tuning সেটআপ/স্কিম দ্রুত ট্রাই করা যায়
  কারণ পুরো pretraining-এর মতো কঠিন বাধা (constraints) থাকে না।

তাহলে কোনটা করা উচিত?

সাধারণ নিয়ম:

- আপনার যদি “বিশাল ডেটা” না থাকে, scratch থেকে ট্রেন করা সাধারণত ভালো আইডিয়া না
- প্রায় সব ক্ষেত্রে **pretrained model** নিয়ে **fine-tune** করাই সেরা পথ

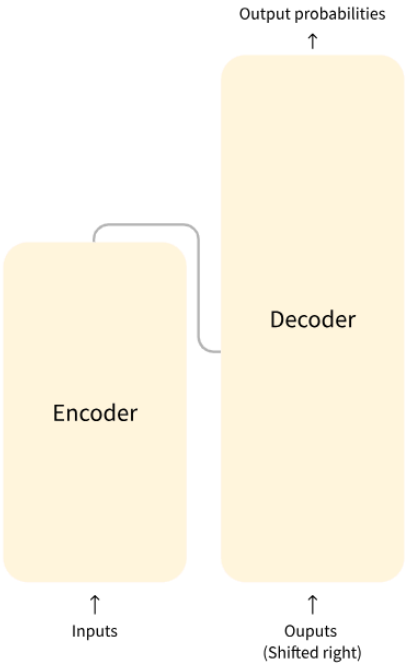
এবং আরও ভালো ফল পেতে:

- এমন pretrained মডেল বেছে নেওয়া উচিত, যেটা আপনার টাস্কের সাথে যতটা সম্ভব “close” (যেমন: মেডিক্যাল টেক্সটের জন্য মেডিক্যাল/বায়োমেড pretrained মডেল)

General Transformer Architecture (Transformer-এর সাধারণ গঠন)

এখন আমরা Transformer-এর “বেসিক স্ট্রাকচার” দেখব। সবকিছু প্রথমবারে ক্লিয়ার না হলেও সমস্যা নেই—পরে প্রতিটি কম্পোনেন্ট আলাদা করে বিস্তারিতভাবে আসবে।

Transformer মডেলকে প্রধানত দুইটা বড় ব্লকে ভাগ করা যায়:



১) Encoder (বাম দিক)

Encoder-এর কাজ হলো:

- ইনপুট গ্রহণ করা
- ইনপুট থেকে একটি শক্তিশালী representation তৈরি করা (features/extracted meaning)

সহজভাবে:
Encoder “বুঝতে” শেখে—ইনপুটে কী বলা হয়েছে, কী মীনিং, কোন অংশ গুরুত্বপূর্ণ ইত্যাদি।

২) Decoder (ডান দিক)

Decoder-এর কাজ হলো:

- Encoder যে representation বানিয়েছে সেটা ব্যবহার করা
- এবং প্রয়োজনীয় অন্য ইনপুট নিয়ে টার্গেট সিকোয়েন্স তৈরি করা (output generation)

সহজভাবে:
Decoder "লিখতে/জেনারেট করতে" শেখে—ইনপুট থেকে আউটপুট বানায়।

Encoder/Decoder আলাদাভাবে কীভাবে ব্যবহার হয়?

Transformer-এর এই দুই অংশকে টাস্ক অনুযায়ী আলাদা করেও ব্যবহার করা যায়।

ক) Encoder-only models

এগুলো ইনপুট বোঝায় বেশি ভালো, তাই ব্যবহার হয়:

- sentence classification
- named entity recognition (NER)
- সাধারণভাবে "understanding" টাইপ কাজ

(উদাহরণ ধরলে BERT-টাইপ মডেল)

খ) Decoder-only models

এগুলো আউটপুট জেনারেশনে শক্তিশালী, তাই ব্যবহার হয়:

- text generation
- গল্প/কোড/রেসপন্স লেখা

(উদাহরণ ধরলে GPT-টাইপ মডেল)

গ) Encoder-Decoder (Sequence-to-Sequence) models

এগুলো এমন জেনারেটিভ টাস্কে ভালো যেখানে ইনপুট থেকে আউটপুট বানাতে হয়:

- translation
- summarization
- text-to-text transformation

(উদাহরণ ধরলে T5-টাইপ মডেল)

Attention Layers (অ্যাটেনশন লেয়ার)

Transformer মডেলের সবচেয়ে গুরুত্বপূর্ণ এবং বৈপ্লবিক অংশ হলো **Attention layer**। এতটাই গুরুত্বপূর্ণ যে Transformer আর্কিটেকচার যেই গবেষণাপত্রে প্রথম প্রকাশিত হয়, তার নামই ছিল—

> **“Attention Is All You Need”**

এই নামটাই ইঙ্গিত দেয়—Transformer মডেলের মূল শক্তি attention-এর মধ্যেই লুকিয়ে আছে।

১) **Attention layer** আসলে কী করে?

সহজ ভাষায় বললে—

> Attention layer মডেলকে শেখায়, একটি শব্দ প্রসেস করার সময় বাক্যের কোন কোন শব্দের দিকে বেশি নজর দিতে হবে, আর কোনগুলো প্রায় উপেক্ষা করা যায়।

মানে, মডেল সব শব্দকে সমান গুরুত্ব দেয় না।
প্রয়োজন অনুযায়ী কিছু শব্দকে বেশি গুরুত্ব **(attention)** দেয়, কিছু শব্দকে কম।

২) কেন **Attention** দরকার? (একটা বাস্তব উদাহরণ)

ধরুন আমাদের কাজ হলো **English → French** অনুবাদ।

ইনপুট বাক্য:

> **“You like this course”**

এখন ভাবুন, মডেল যখন **“like”** শব্দটা অনুবাদ করতে চাইবে:

- French ভাষায় verb-এর রূপ subject অনুযায়ী বদলায়
- তাই “like” অনুবাদ করতে হলে মডেলকে **“You”** শব্দটার দিকে তাকাতেই হবে
- বাক্যের বাকি শব্দগুলো (“this”, “course”) এই মুহূর্তে তেমন কাজে আসে না

অর্থাৎ:

- “like” → “You”-এর উপর নির্ভরশীল

আবার ধরুন, মডেল যখন **“this”** শব্দটা অনুবাদ করবে:

- French ভাষায় “this” আলাদা হয় noun masculine না feminine তার উপর

- তাই “this” অনুবাদ করতে হলে মডেলকে **“course”** শব্দটার দিকে তাকাতে হবে
- বাক্যের অন্য শব্দগুলো তখন তেমন দরকার নেই

এখানে আমরা দেখছি:

- আলাদা শব্দ → আলাদা context দরকার
- Attention layer এই context নির্বাচন করতেই সাহায্য করে

৩) দূরের শব্দের দিকেও নজর দিতে হয়

বাস্তব ভাষা আরও জটিল।

অনেক সময়:

- যে শব্দটার উপর নির্ভর করছে অর্থ
- সেটা বাক্যের অনেক আগে বা অনেক পরে থাকতে পারে

Attention layer মডেলকে এই ক্ষমতা দেয়—

- বাক্যের যেকোন জায়গার শব্দের সাথে সম্পর্ক তৈরি করার

এটাই Transformer-কে RNN বা LSTM-এর চেয়ে অনেক শক্তিশালী করেছে।

৪) **Attention** শুধু অনুবাদে নয়—সব **NLP** টাস্কেই

এই ধারণাটা শুধু translation-এর জন্য না।

যেকোনো NLP টাস্কে:

- একটি শব্দের নিজস্ব অর্থ আছে
- কিন্তু সেই অর্থ **context** অনুযায়ী বদলে যায়

Context হতে পারে:

- আগের শব্দ
- পরের শব্দ
- বা একাধিক শব্দ একসাথে

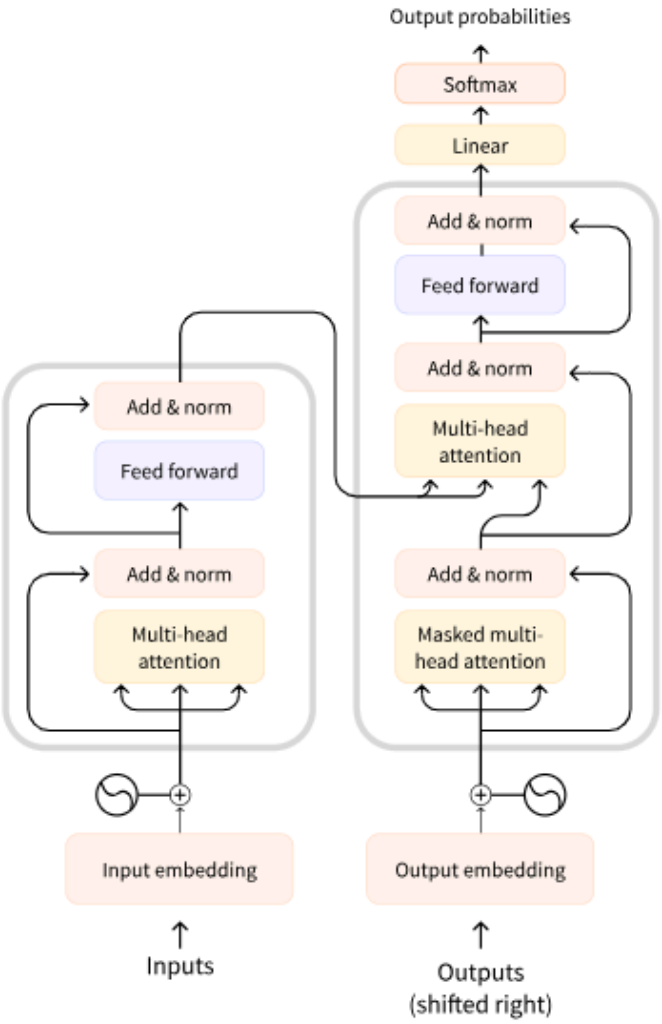
Attention layer এই context ধরার কাজটাই করে।

Transformer-এর মূল আর্কিটেকচার (Original Architecture)

Transformer আর্কিটেকচার মূলত **translation** টাস্কের জন্য ডিজাইন করা হয়েছিল।

এখানে দুইটা বড় অংশ থাকে:

- **Encoder**
- **Decoder**



৫) **Encoder** কীভাবে কাজ করে?

Training-এর সময়:

- Encoder ইনপুট ভাষার পুরো বাক্য পায় (যেমন English sentence)

Encoder-এর attention layer:

- বাক্যের সব শব্দের দিকে তাকাতে পারে
- কারণ কোনো শব্দের অর্থ আগের ও পরের—দুই দিকের শব্দের উপরই নির্ভর করতে পারে

Encoder-এর মূল কাজ:

> ইনপুট বাক্যটাকে ভালোভাবে "বোঝা" এবং তার একটা rich representation বানানো

৬) **Decoder** কীভাবে কাজ করে?

Decoder-এর কাজ হলো:

- Encoder-এর বোঝা তথ্য ব্যবহার করে
- লক্ষ্য ভাষায় (target language) বাক্য তৈরি করা

কিন্তু Decoder একটা গুরুত্বপূর্ণ নিয়ম মেনে চলে:

> Decoder একসাথে পুরো বাক্য দেখে না
> এটা ধাপে ধাপে শব্দ তৈরি করে

কেন?

কারণ আমরা চাই না মডেল ভবিষ্যৎ শব্দ দেখে ফেলে।

উদাহরণ:

- যদি মডেল ৪ নম্বর শব্দ প্রেডিক্ট করার সময়
- আগে থেকেই ৪ নম্বর শব্দ দেখে ফেলে
  তাহলে শেখার কাজটা আর কঠিন থাকবে না (cheating হয়ে যাবে)

৭) **Decoder-**এর **Attention** কীভাবে সীমাবদ্ধ করা হয়?

Training-এর সময়:

- Decoder-কে পুরো target sentence দেওয়া হয় (speed বাড়ানোর জন্য)
- কিন্তু **attention mask** ব্যবহার করে এমনভাবে ব্লক করা হয় যেন:

  ৪ নম্বর শব্দ প্রেডিক্ট করার সময়
  Decoder শুধু ১–৩ নম্বর শব্দই দেখতে পায়

এটাকে বলা হয়:

- **Causal / Masked Attention**

৮) Decoder-এর দুই ধরনের Attention layer

একটা Decoder ব্লকের ভেতরে সাধারণত দুইটা attention layer থাকে:

১) Self-Attention (Masked)

- Decoder-এর আগের তৈরি করা শব্দগুলোর দিকে তাকায়
- ভবিষ্যৎ শব্দ দেখতে পায় না

২) Encoder–Decoder Attention

- Encoder-এর আউটপুটের দিকে তাকায়
- ফলে পুরো ইনপুট বাক্যের context ব্যবহার করতে পারে

এটা খুব গুরুত্বপূর্ণ কারণ:

- বিভিন্ন ভাষায় শব্দের order আলাদা
- অনেক সময় বাক্যের শেষে থাকা তথ্য শুরুতে অনুবাদ করার সময় দরকার হয়

৯) Attention Mask আর Padding

বাস্তবে আমরা অনেক বাক্য একসাথে (batch) প্রসেস করি।
কিন্তু সব বাক্যের দৈর্ঘ্য সমান হয় না।

তাই:

- ছোট বাক্যে **padding token** যোগ করা হয়

Attention mask ব্যবহার করে:

- মডেলকে বলা হয় padding token-গুলোর দিকে যেন attention না দেয়

Architecture বনাম Checkpoint বনাম Model

এখানে তিনটা শব্দ প্রায়ই গুলিয়ে যায়, তাই পরিষ্কার করে বলি।

Architecture

- মডেলের কাঠামো
- কোন লেয়ার থাকবে, কীভাবে কানেক্ট হবে

উদাহরণ:

- BERT architecture
- GPT architecture

Checkpoint

- ওই architecture-এর জন্য ট্রেন করা **weights**
- মানে শেখা জ্ঞান

উদাহরণ:

- `bert-base-cased`
- `bert-base-uncased`

Model

- একটা সাধারণ শব্দ
- অনেক সময় architecture + checkpoint—দুটোকেই বোঝাতে ব্যবহার হয়

উদাহরণ:

- “BERT model” বলা হলে
    - architecture-ও বোঝাতে পারে
    - নির্দিষ্ট checkpoint-ও বোঝাতে পারে
      কনটেক্সট দেখে বুঝতে হয়