Transformers II: BERT, GPT, Pretraining, Finetuning and Generation

Same block, different mask + objective + head ⟹ different model family

Prof. Nipun Batra

ES 667 · Deep Learning · IIT Gandhinagar

One block, three families

Where we are — recall Transformers I

Last lecture built the **Transformer block**: self-attention over all tokens, no recurrence.

Where we are — recall Transformers I

Last lecture built the **Transformer block**: self-attention over all tokens, no recurrence.

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

Where we are — recall Transformers I

Last lecture built the **Transformer block**: self-attention over all tokens, no recurrence.

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}} + \text{mask}\right) V$$

Where we are — recall Transformers I

Last lecture built the **Transformer block**: self-attention over all tokens, no recurrence.

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}} + \text{mask}\right) V$$

One block = multi-head self-attention → residual + norm → FFN → residual + norm. This lecture: the **same block** becomes three different model **families**.

The whole lecture in one line

We keep the Transformer block **fixed** and change three knobs:

The whole lecture in one line

We keep the Transformer block **fixed** and change three knobs:

$$\textbf{mask} + \textbf{objective} + \textbf{head} \Rightarrow \textbf{model family}$$

The whole lecture in one line

We keep the Transformer block **fixed** and change three knobs:

mask + **objective** + **head** ⇒ **model family**

Knob	**Controls**	**Choices**
attention mask	what a token may **see**	bidirectional / causal / cross
pretraining objective	what the model **learns to do**	masked / next-token / denoise
output head	how we **read it out**	`[CLS]` / LM head / decoder

The whole lecture in one line

We keep the Transformer block **fixed** and change three knobs:

mask + **objective** + **head** ⇒ **model family**

Knob	**Controls**	**Choices**
attention mask	what a token may **see**	bidirectional / causal / cross
pretraining objective	what the model **learns to do**	masked / next-token / denoise
output head	how we **read it out**	`[CLS]` / LM head / decoder

architecture controls information flow; objective controls what it learns

The three families

V VISUAL

	Encoder-only	Decoder-only	Encoder–decoder
mask	bidirectional	causal	enc: bidir · dec: causal
objective	masked-token	next-token	seq2seq / denoise
reads	both sides	left context	source $\rightarrow$ target
best at	understanding	generation	transduction
example	**BERT**	**GPT**	**T5 · BART**

The three families

V VISUAL

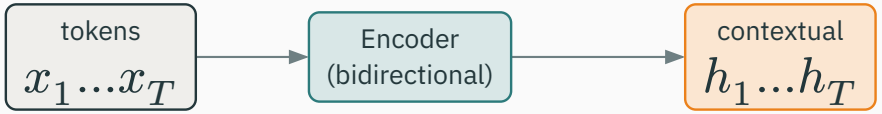
	Encoder-only	Decoder-only	Encoder–decoder
mask	bidirectional	causal	enc: bidir · dec: causal
objective	masked-token	next-token	seq2seq / denoise
reads	both sides	left context	source → target
best at	understanding	generation	transduction
example	**BERT**	**GPT**	**T5 · BART**

three columns, one underlying block — the differences are the **mask**, the **objective** and the **head**

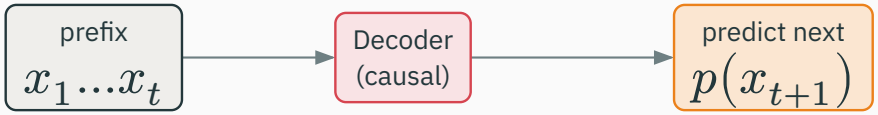
One picture, three families

V VISUAL

encoder-only



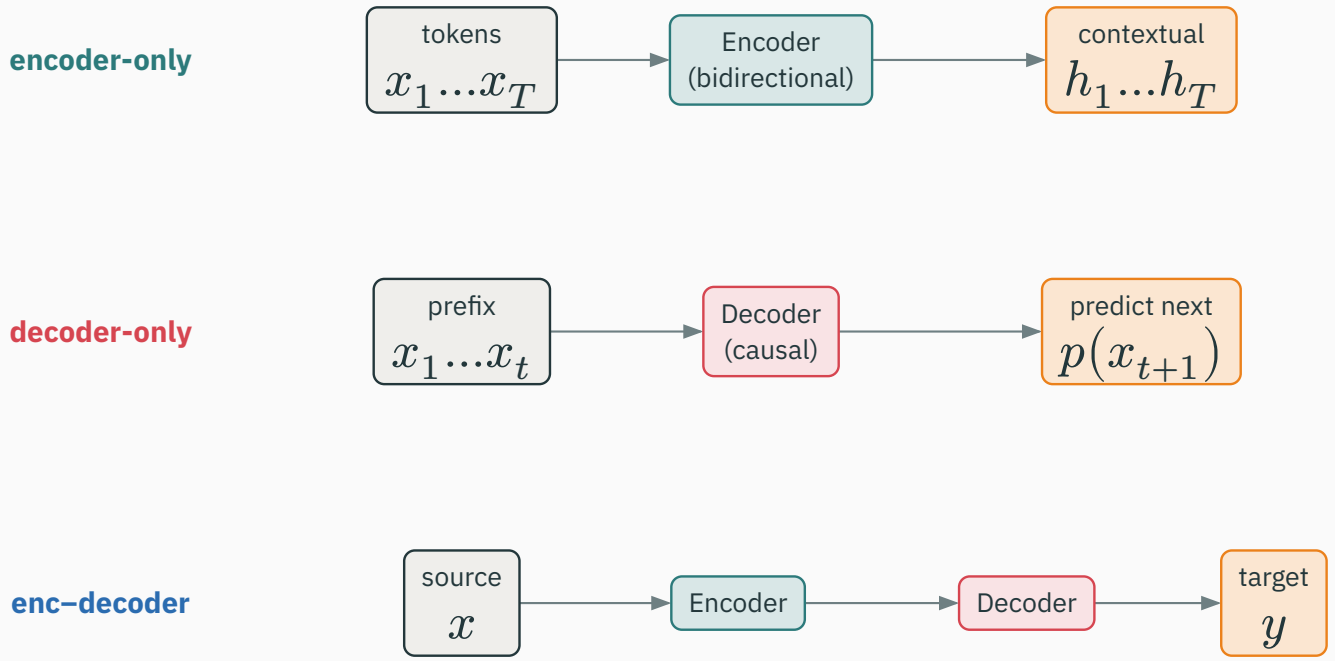
decoder-only



enc–decoder

One picture, three families

V VISUAL



encoder-only reads a whole sequence into **contextual reps**; decoder-only **predicts the next token**; encoder–decoder maps a **source** to a **target**

The masks that define them

V VISUAL

The mask is the single biggest lever — it decides who may attend to whom:

The masks that define them

V VISUAL

The mask is the single biggest lever — it decides who may attend to whom:

bidirectional (encoder)

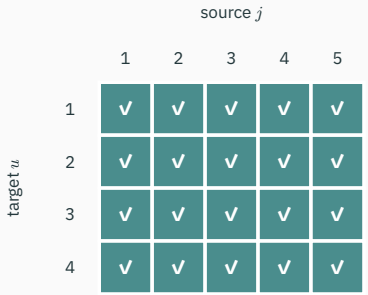
BERT — no mask

causal (decoder)

GPT — mask future

cross-attention

decoder ⟶ encoder



The masks that define them

V VISUAL

The mask is the single biggest lever — it decides who may attend to whom:

bidirectional (encoder)

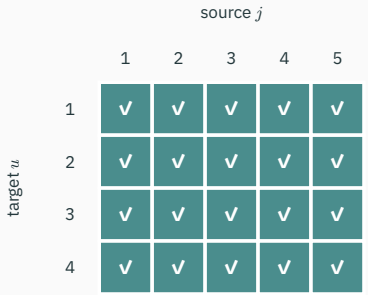
BERT — no mask

causal (decoder)

GPT — mask future

cross-attention

decoder ⟶ encoder



bidirectional (BERT): all-to-all · causal (GPT): only the past · cross-attention: decoder reads encoder

Interactive: model-family masks

I INTERACTIVE

INTERACTIVE

↗ nipunbatra.github.io/interactive-articles/attention

Toggle between **bidirectional**, **causal** and **cross-attention** masks on a short sequence and watch which entries of the $T \times T$ attention matrix light up — the same operation, three information-flow patterns.

Encoder-only: BERT

The encoder-only Transformer

D DERIVATION

Feed all tokens in at once; every token attends **every** non-padding token

The encoder-only Transformer

D DERIVATION

no causal mask:

The encoder-only Transformer

D DERIVATION

no causal mask:

$$x_1, ..., x_T \xrightarrow{\text{Encoder}} h_1, ..., h_T$$

The encoder-only Transformer

D DERIVATION

no causal mask:

$$x_1, \ldots, x_T \xrightarrow{\text{Encoder}} h_1, \ldots, h_T$$

- each h_t is a **contextual representation** of token t — informed by the whole sequence;
- no left-to-right restriction: position t may look **left and right**.

The encoder-only Transformer

D DERIVATION

no causal mask:

$$x_1, \ldots, x_T \xrightarrow{\text{Encoder}} h_1, \ldots, h_T$$

- each h_t is a **contextual representation** of token t — informed by the whole sequence;
- no left-to-right restriction: position t may look **left and right**.

the output is not a prediction — it is a **representation per token** to feed a task head

What BERT is good for

V VISUAL

Encoder-only models shine at **understanding** — mapping text to a fixed target:

What BERT is good for

V VISUAL

Encoder-only models shine at **understanding** — mapping text to a fixed target:

Task	**Shape**
sentence classification / sentiment	$x_{1:T} \to y$
natural language inference (NLI)	pair $\to$ {entail, contradict, neutral}
extractive QA	passage $\to$ answer span
token classification / NER	$x_{1:T} \to y_{1:T}$
retrieval / embeddings	text $\to$ vector

What BERT is good for

V VISUAL

Encoder-only models shine at **understanding** — mapping text to a fixed target:

Task	**Shape**
sentence classification / sentiment	$x_{1:T} \to y$
natural language inference (NLI)	pair $\to$ {entail, contradict, neutral}
extractive QA	passage $\to$ answer span
token classification / NER	$x_{1:T} \to y_{1:T}$
retrieval / embeddings	text $\to$ vector

whenever the output is a **label** or a **span**, both-side context helps

Why not next-token LM?

D DERIVATION

A causal LM only ever sees the **left**:

Why not next-token LM?

D DERIVATION

A causal LM only ever sees the **left**:

$p(x_t \mid x_{<t})$ (left-to-right only)

Why not next-token LM?

D DERIVATION

A causal LM only ever sees the **left**:

$p(x_t \mid x_{<t})$ (left-to-right only)

But **understanding** a word often needs **both sides**:

"The **bank** approved the loan" vs "The **bank** was flooded"

Why not next-token LM?

D DERIVATION

A causal LM only ever sees the **left**:

$p(x_t \mid x_{<t})$ (left-to-right only)

But **understanding** a word often needs **both sides**:

"The **bank** approved the loan" vs "The **bank** was flooded"

The word after bank disambiguates it. A left-only model has not seen it yet when it encodes bank. Bidirectional context lets the representation use **the whole sentence**.

Masked language modeling (MLM)

D DERIVATION

BERT’s trick: **corrupt** some tokens, then predict the originals from **both sides**:

Masked language modeling (MLM)

D DERIVATION

BERT's trick: **corrupt** some tokens, then predict the originals from **both sides**:

`the cat sat on the [MASK]` $\longrightarrow$ predict `mat`

Masked language modeling (MLM)

D DERIVATION

BERT's trick: **corrupt** some tokens, then predict the originals from **both sides**:

`the cat sat on the [MASK]` $\longrightarrow$ predict `mat`

$$p_\theta\left(x_m \mid x_{\setminus m}\right)$$ (recover masked token m from all the others)

Masked language modeling (MLM)

D DERIVATION

BERT's trick: **corrupt** some tokens, then predict the originals from **both sides**:

`the cat sat on the [MASK]` $\longrightarrow$ predict `mat`

$p_\theta\left(x_m \mid x_{\setminus m}\right)$ (recover masked token m from all the others)

Loss is applied **only on the masked positions** M:

$$\mathcal{L}_{\text{MLM}} = -\sum_{m \in M} \log p_\theta(x_m \mid \tilde{x})$$

Masked language modeling (MLM)

D DERIVATION

BERT's trick: **corrupt** some tokens, then predict the originals from **both sides**:

`the cat sat on the [MASK]` $\longrightarrow$ predict `mat`

$p_\theta\left(x_m \mid x_{\setminus m}\right)$ (recover masked token m from all the others)

Loss is applied **only on the masked positions** M:

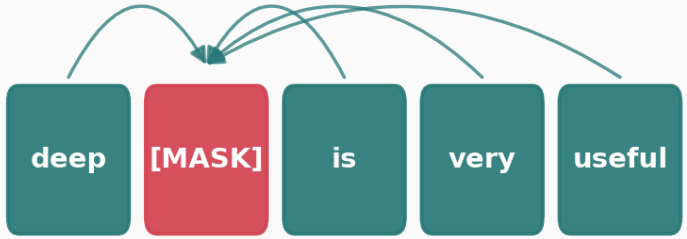
$$\mathcal{L}_{\text{MLM}} = -\sum_{m \in M} \log p_\theta(x_m \mid \tilde{x})$$

$\tilde{x}$ is the corrupted sequence · unmasked tokens get **no** loss, they are only context

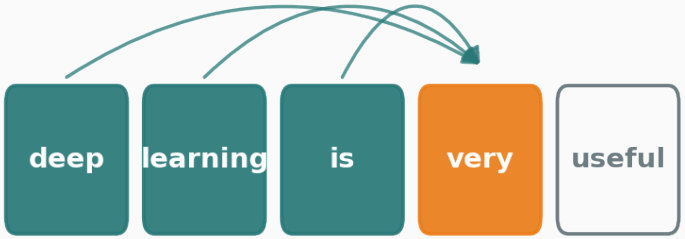
MLM vs next-token, side by side

V VISUAL

MLM (BERT): predict [MASK] from both sides



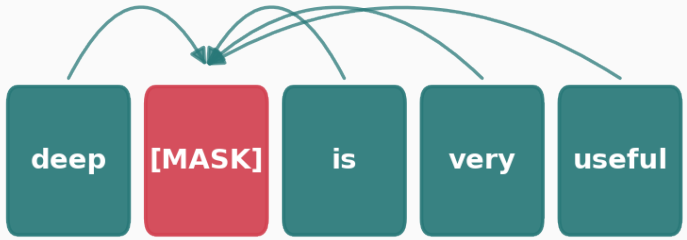
CLM (GPT): predict next token from the left only



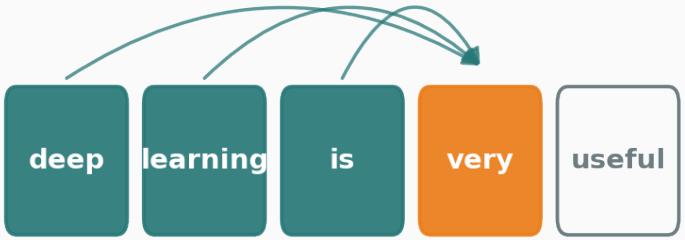
MLM vs next-token, side by side

V VISUAL

MLM (BERT): predict [MASK] from both sides



CLM (GPT): predict next token from the left only



MLM predicts a hidden token from **both** neighbours; a causal LM predicts the **next** token from the **left** only

MLM worked example

D DERIVATION

Sequence `deep learning is useful`; mask position 2:

MLM worked example

D DERIVATION

Sequence `deep learning is useful`; mask position 2:

`deep [MASK] is useful` target: `learning`

MLM worked example

D DERIVATION

Sequence `deep learning is useful`; mask position 2:

```
deep [MASK] is useful
```

target: `learning`

The encoder sees `deep`, `is`, `useful` (both sides) and scores the vocabulary at position 2:

$$\mathcal{L} = -\log p_\theta(\text{learning} \mid \tilde{x})$$

MLM worked example

D DERIVATION

Sequence `deep learning is useful`; mask position 2:

`deep [MASK] is useful` target: `learning`

The encoder sees `deep`, `is`, `useful` (both sides) and scores the vocabulary at position 2:

$$\mathcal{L} = -\log p_\theta(\text{learning} \mid \tilde{x})$$

a single V-way classification at the masked slot — using **right** context (`is useful`) too

BERT input format

D DERIVATION

Two sentences packed into one sequence with special tokens:

BERT input format

D DERIVATION

Two sentences packed into one sequence with special tokens:

`[CLS]` sentence A `[SEP]` sentence B `[SEP]`

BERT input format

D DERIVATION

Two sentences packed into one sequence with special tokens:

```
[CLS] sentence A [SEP] sentence B [SEP]
```

Each token's input embedding is a **sum of three** embeddings:

$$r_t = e_t + p_t + s_t$$

BERT input format

D DERIVATION

Two sentences packed into one sequence with special tokens:

```
[CLS] sentence A [SEP] sentence B [SEP]
```

Each token's input embedding is a **sum of three** embeddings:

$$r_t = e_t + p_t + s_t$$

Component	Symbol	Encodes
token	e_t	which word / subword
position	p_t	where it sits
segment	s_t	sentence A vs B

BERT input format

D DERIVATION

Two sentences packed into one sequence with special tokens:

```
[CLS] sentence A [SEP] sentence B [SEP]
```

Each token's input embedding is a **sum of three** embeddings:

$$r_t = e_t + p_t + s_t$$

Component	Symbol	Encodes
token	e_t	which word / subword
position	p_t	where it sits
segment	s_t	sentence A vs B

the two-sentence format feeds a second pre-training task, **Next-Sentence Prediction** (NSP): from `[CLS]`, predict whether B really follows A

The [CLS] representation

D DERIVATION

A special [CLS] token sits at the front; its final state summarizes the **whole** sequence:

The [CLS] representation

D DERIVATION

A special [CLS] token sits at the front; its final state summarizes the **whole** sequence:

$$h_{[\mathrm{CLS}]} \xrightarrow{\text{classifier}} p(y \mid x) = \mathrm{softmax}\big(W\, h_{[\mathrm{CLS}]} + b\big)$$

The [CLS] representation

D DERIVATION

A special [CLS] token sits at the front; its final state summarizes the **whole** sequence:

$$h_{[\mathrm{CLS}]} \xrightarrow{\text{classifier}} p(y \mid x) = \mathrm{softmax}\big(W\, h_{[\mathrm{CLS}]} + b\big)$$

Attention lets [CLS] gather from every token, so its representation is a natural **pooled** summary. A small head on top of $h_{[\mathrm{CLS}]}$ gives sentence-level predictions.

The [CLS] representation

D DERIVATION

A special [CLS] token sits at the front; its final state summarizes the **whole** sequence:

$$h_{[\mathrm{CLS}]} \xrightarrow{\text{classifier}} p(y \mid x) = \mathrm{softmax}\big(W\, h_{[\mathrm{CLS}]} + b\big)$$

Attention lets [CLS] gather from every token, so its representation is a natural **pooled** summary. A small head on top of $h_{[\mathrm{CLS}]}$ gives sentence-level predictions.

classification, NLI, sentiment — all read out from this one vector

Token classification

D DERIVATION

Some tasks want a label at **every** position — put the head on **each** h_t:

Token classification

D DERIVATION

Some tasks want a label at **every** position — put the head on **each** h_t:

$$p(y_t \mid x) = \mathrm{softmax}(W\, h_t + b)$$ (one prediction per token)

Token classification

D DERIVATION

Some tasks want a label at **every** position — put the head on **each** h_t:

$$p(y_t \mid x) = \mathrm{softmax}(W\, h_t + b) \qquad \text{(one prediction per token)}$$

- named-entity recognition, part-of-speech tagging;
- a **padding mask** keeps padded positions out of the loss.

Token classification

D DERIVATION

Some tasks want a label at **every** position — put the head on **each** h_t:

$$p(y_t \mid x) = \mathrm{softmax}(W\, h_t + b) \qquad \text{(one prediction per token)}$$

- named-entity recognition, part-of-speech tagging;
- a **padding mask** keeps padded positions out of the loss.

bidirectional context helps: tagging a word can use the words **after** it

Extractive QA

D DERIVATION

Answer = a **span** in the passage. Learn a **start** and an **end** pointer over tokens:

Extractive QA

D DERIVATION

Answer = a **span** in the passage. Learn a **start** and an **end** pointer over tokens:

$$p_{\text{start}(t)} = \text{softmax}(w_s^\top h_t), \qquad p_{\text{end}(t)} = \text{softmax}(w_e^\top h_t)$$

Extractive QA

D DERIVATION

Answer = a **span** in the passage. Learn a **start** and an **end** pointer over tokens:

$$p_{\text{start}(t)} = \text{softmax}\big(w_s^\top h_t\big), \qquad p_{\text{end}(t)} = \text{softmax}\big(w_e^\top h_t\big)$$

Train to point at the true start s^* and end e^*:

$$\mathcal{L} = -\log p_{\text{start}(s^*)} - \log p_{\text{end}(e^*)}$$

Extractive QA

D DERIVATION

Answer = a **span** in the passage. Learn a **start** and an **end** pointer over tokens:

$$p_{\text{start}(t)} = \text{softmax}\big(w_s^\top h_t\big), \qquad p_{\text{end}(t)} = \text{softmax}\big(w_e^\top h_t\big)$$

Train to point at the true start s^* and end e^*:

$$\mathcal{L} = -\log p_{\text{start}(s^*)} - \log p_{\text{end}(e^*)}$$

two little heads over the same contextual states — no text is generated, just **located**

The BERT finetuning pattern

V VISUAL

Every downstream task is the **same** recipe:

The BERT finetuning pattern

V VISUAL

Every downstream task is the **same** recipe:

pretrained encoder $\rightarrow$ task head $\rightarrow$ supervised loss

The BERT finetuning pattern

V VISUAL

Every downstream task is the **same** recipe:

$$\text{pretrained encoder} \;\rightarrow\; \text{task head} \;\rightarrow\; \text{supervised loss}$$

Start from the MLM-pretrained weights; bolt on a small task head ($h_{[\mathrm{CLS}]}$ or per-token); train on labelled data. Typically **all** parameters update.

The BERT finetuning pattern

V VISUAL

Every downstream task is the **same** recipe:

pretrained encoder $\rightarrow$ task head $\rightarrow$ supervised loss

Start from the MLM-pretrained weights; bolt on a small task head ($h_{[\mathrm{CLS}]}$ or per-token); train on labelled data. Typically **all** parameters update.

one pretrained backbone, many cheap heads — the pretrain-then-finetune paradigm

RoBERTa: the recipe matters

D DERIVATION

Same architecture and MLM objective as BERT — but a **better training recipe**:

RoBERTa: the recipe matters

D DERIVATION

Same architecture and MLM objective as BERT — but a **better training recipe**:

Changes — more data, longer training, bigger batches, dynamic masking; dropped the next-sentence task.

Result — a large jump on the same benchmarks, with **no** architectural change.

RoBERTa: the recipe matters

D DERIVATION

Same architecture and MLM objective as BERT — but a **better training recipe**:

Changes — more data, longer training, bigger batches, dynamic masking; dropped the next-sentence task.

Result — a large jump on the same benchmarks, with **no** architectural change.

the objective matters, but so does the recipe: data, scale, and training choices

Decoder-only: GPT

The decoder-only Transformer

D DERIVATION

Causal self-attention: token t may attend only to $x_{\leq t}$, and predicts the next token:

The decoder-only Transformer

D DERIVATION

Causal self-attention: token t may attend only to $x_{\leq t}$, and predicts the next token:

$$h_t = f(x_{\leq t}), \qquad p(x_{t+1} \mid x_{\leq t}) = \mathrm{softmax}(W_o\, h_t)$$

The decoder-only Transformer

D DERIVATION

Causal self-attention: token t may attend only to $x_{\leq t}$, and predicts the next token:

$$h_t = f(x_{\leq t}), \qquad p(x_{t+1} \mid x_{\leq t}) = \mathrm{softmax}(W_o\, h_t)$$

Drop the encoder and cross-attention from the Transformer: what remains is a stack of **causal self-attention + FFN** blocks, capped by an LM head.

The decoder-only Transformer

D DERIVATION

Causal self-attention: token t may attend only to $x_{\le t}$, and predicts the next token:

$$h_t = f(x_{\le t}), \qquad p(x_{t+1} \mid x_{\le t}) = \mathrm{softmax}(W_o\, h_t)$$

Drop the encoder and cross-attention from the Transformer: what remains is a stack of **causal self-attention + FFN** blocks, capped by an LM head.

this is the **same** next-token model of Lecture 11 — now with attention instead of a fixed window

The language-modeling objective

D DERIVATION

Factorize the sequence by the chain rule, then take negative log-likelihood:

The language-modeling objective

D DERIVATION

Factorize the sequence by the chain rule, then take negative log-likelihood:

$$p(x_{1:T}) = \prod_{t=1}^{T} p(x_t \mid x_{<t})$$

The language-modeling objective

D DERIVATION

Factorize the sequence by the chain rule, then take negative log-likelihood:

$$p(x_{1:T}) = \prod_{t=1}^{T} p(x_t \mid x_{<t})$$

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

The language-modeling objective

D DERIVATION

Factorize the sequence by the chain rule, then take negative log-likelihood:

$$p(x_{1:T}) = \prod_{t=1}^{T} p(x_t \mid x_{<t})$$

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

token-level cross-entropy — **every** position is a training example, computed in parallel

The causal mask

D DERIVATION

Compute scores, then forbid attending to the future **before** the softmax:

The causal mask

D DERIVATION

Compute scores, then forbid attending to the future **before** the softmax:

$$A_{ij} = 0 \quad \text{for} \quad j > i$$

The causal mask

D DERIVATION

Compute scores, then forbid attending to the future **before** the softmax:

$$A_{ij} = 0 \quad \text{for} \quad j > i$$

Set $S_{ij} = -\infty$ when $j > i$; softmax turns those into 0. Each row normalizes over the **past and present only** — no token can peek at its own target.

The causal mask

D DERIVATION

Compute scores, then forbid attending to the future **before** the softmax:

$$A_{ij} = 0 \quad \text{for} \quad j > i$$

Set $S_{ij} = -\infty$ when $j > i$; softmax turns those into 0. Each row normalizes over the **past and present only** — no token can peek at its own target.

this lower-triangular pattern is **the** difference between GPT and BERT attention

Shifted labels

D DERIVATION

Feed the sequence in; the target is the **same sequence shifted left by one**:

Shifted labels

D DERIVATION

Feed the sequence in; the target is the **same sequence shifted left by one**:

position t	1	2	3	4
input x_t	<BOS>	deep	learning	is
target x_{t+1}	deep	learning	is	useful

Shifted labels

D DERIVATION

Feed the sequence in; the target is the **same sequence shifted left by one**:

position t	1	2	3	4
input x_t	<BOS>	deep	learning	is
target x_{t+1}	deep	learning	is	useful

the causal mask makes this safe — position t predicts $t+1$ without seeing it

GPT-style pretraining

D DERIVATION

Pretrain the decoder to predict the **next token** on a large corpus. Then adapt:

GPT-style pretraining

D DERIVATION

Pretrain the decoder to predict the **next token** on a large corpus. Then adapt:

Adaptation	**What it means**
prompting	condition on text; read the continuation
finetuning	continue training on task data
instruction tuning	finetune on (instruction $\rightarrow$ response) pairs
RLHF (later)	align to human preferences

GPT-style pretraining

D DERIVATION

Pretrain the decoder to predict the **next token** on a large corpus. Then adapt:

Adaptation	**What it means**
prompting	condition on text; read the continuation
finetuning	continue training on task data
instruction tuning	finetune on (instruction $\rightarrow$ response) pairs
RLHF (later)	align to human preferences

GPT-1 introduced **generative pretrain + finetune**; the objective never changes — only the adaptation

GPT-2: LM as unsupervised multitask

D DERIVATION

Train a **large** decoder-only model on broad web text — then ask it to do tasks with **no** finetuning:

GPT-2: LM as unsupervised multitask

D DERIVATION

Train a **large** decoder-only model on broad web text — then ask it to do tasks with **no** finetuning:

A big enough next-token model performs many tasks **zero-shot**, purely from a prompt. Language modeling on diverse text is **implicitly** multitask learning (Radford et al.).

GPT-2: LM as unsupervised multitask

D DERIVATION

Train a **large** decoder-only model on broad web text — then ask it to do tasks with **no** finetuning:

A big enough next-token model performs many tasks **zero-shot**, purely from a prompt. Language modeling on diverse text is **implicitly** multitask learning (Radford et al.).

scale + broad data $\Rightarrow$ a single LM that generalizes to unseen tasks from prompts alone

Prompting as conditioning

D DERIVATION

Prompting is **not** an architecture change — it is just **conditioning the LM**:

Prompting as conditioning

D DERIVATION

Prompting is **not** an architecture change — it is just **conditioning the LM**:

$$p(\text{continuation} \mid \text{prompt})$$

Prompting as conditioning

D DERIVATION

Prompting is **not** an architecture change — it is just **conditioning the LM**:

$$p(\text{continuation} \mid \text{prompt})$$

```
Translate to French: cat =>  (the model continues)  chat
```

Prompting as conditioning

D DERIVATION

Prompting is **not** an architecture change — it is just **conditioning the LM**:

$$p(\text{continuation} \mid \text{prompt})$$

```
Translate to French: cat =>  (the model continues)  chat
```

the prompt sets the left context; the **same** next-token machinery produces the answer

Few-shot prompting

D DERIVATION

Put a few **demonstrations** directly in the context, then the query:

Few-shot prompting

D DERIVATION

Put a few **demonstrations** directly in the context, then the query:

```
great movie => positive
awful plot => negative
loved every minute => ?
```

Few-shot prompting

D DERIVATION

Put a few **demonstrations** directly in the context, then the query:

```
great movie => positive
awful plot => negative
loved every minute => ?
```

No weights change. The model **reads the pattern** from the examples in its context window and continues it — “in-context learning” (GPT-3; Brown et al., 2020).

Interactive: in-context learning

I INTERACTIVE

INTERACTIVE

↗ nipunbatra.github.io/interactive-articles/in-context-learning

Add demonstrations to a prompt and watch a decoder-only model shift its next-token distribution toward the pattern you set — few-shot prompting with **no** gradient steps.

Classification via prompting

D DERIVATION

Recast a label task as **next-token scoring**:

Classification via prompting

D DERIVATION

Recast a label task as **next-token scoring**:

```
Review: a dull, lifeless film. Sentiment: → negative
```

Classification via prompting

D DERIVATION

Recast a label task as **next-token scoring**:

```
Review: a dull, lifeless film. Sentiment: → negative
```

Compare the LM's probability of each candidate label token:

$$\hat{y} = \arg\max_{y \in \{\text{positive, negative}\}} p_\theta(y \mid \text{prompt})$$

Classification via prompting

D DERIVATION

Recast a label task as **next-token scoring**:

```
Review: a dull, lifeless film. Sentiment: ⟶ negative
```

Compare the LM's probability of each candidate label token:

$$\hat{y} = \arg\max_{y \in \{\text{positive, negative}\}} p_\theta(y \mid \text{prompt})$$

the classifier **is** the language model — pick the label it finds most probable

The generation loop

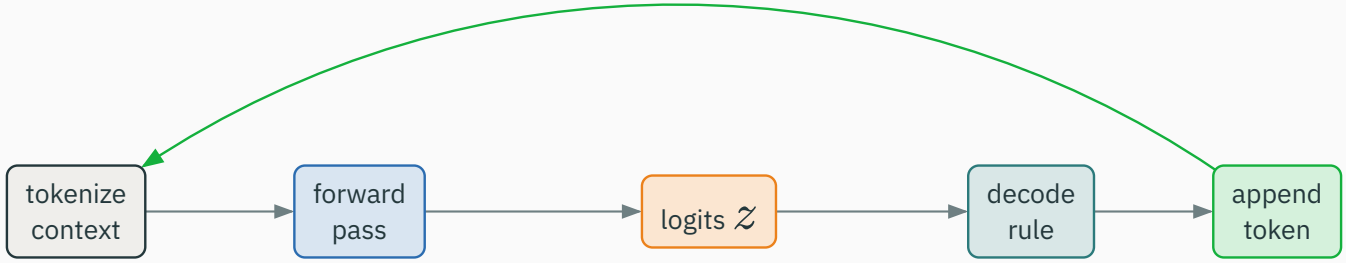
V VISUAL

Decoder-only generation is one loop, run token by token:

The generation loop

V VISUAL

Decoder-only generation is one loop, run token by token:

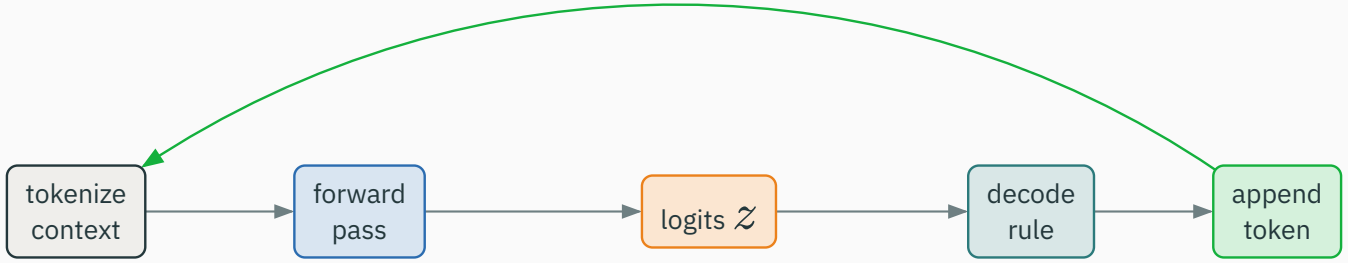


repeat until <EOS> or max length

The generation loop

V VISUAL

Decoder-only generation is one loop, run token by token:



tokenize ⟶ forward ⟶ logits ⟶ **decode rule** ⟶ append ⟶ repeat (the rule is Part VII)

Encoder–decoder

The architecture

D DERIVATION

Two stacks: an encoder reads the **source**, a decoder generates the **target**:

The architecture

D DERIVATION

Two stacks: an encoder reads the **source**, a decoder generates the **target**:

$$H = \mathrm{Encoder}(x), \qquad p(y_u \mid y_{<u}, x) = \mathrm{softmax}(W_o\, s_u)$$

The architecture

D DERIVATION

Two stacks: an encoder reads the **source**, a decoder generates the **target**:

$$H = \mathrm{Encoder}(x), \qquad p(y_u \mid y_{<u}, x) = \mathrm{softmax}(W_o\, s_u)$$

The decoder layer has **three** sublayers:

1. **causal** self-attention over the target prefix $y_{<u}$;
2. **cross-attention** into the encoder states H;
3. a position-wise FFN.

The architecture

D DERIVATION

Two stacks: an encoder reads the **source**, a decoder generates the **target**:

$$H = \mathrm{Encoder}(x), \qquad p(y_u \mid y_{<u}, x) = \mathrm{softmax}(W_o\, s_u)$$

The decoder layer has **three** sublayers:

1. **causal** self-attention over the target prefix $y_{<u}$;
2. **cross-attention** into the encoder states H;
3. a position-wise FFN.

self-attention reads what's generated so far; cross-attention reads the **source**

Cross-attention

D DERIVATION

The bridge between the stacks: **queries from the decoder**, **keys and values from the encoder**:

Cross-attention

D DERIVATION

The bridge between the stacks: **queries from the decoder**, **keys and values from the encoder**:

$$Q = SW_Q \text{ (decoder)}, \quad K = HW_K, V = HW_V \text{ (encoder)}$$

Cross-attention

D DERIVATION

The bridge between the stacks: **queries from the decoder**, **keys and values from the encoder**:

$$Q = SW_Q \text{ (decoder)}, \quad K = HW_K, V = HW_V \text{ (encoder)}$$

$$\text{CrossAttn}(S, H) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right) V$$

Cross-attention

D DERIVATION

The bridge between the stacks: **queries from the decoder**, **keys and values from the encoder**:

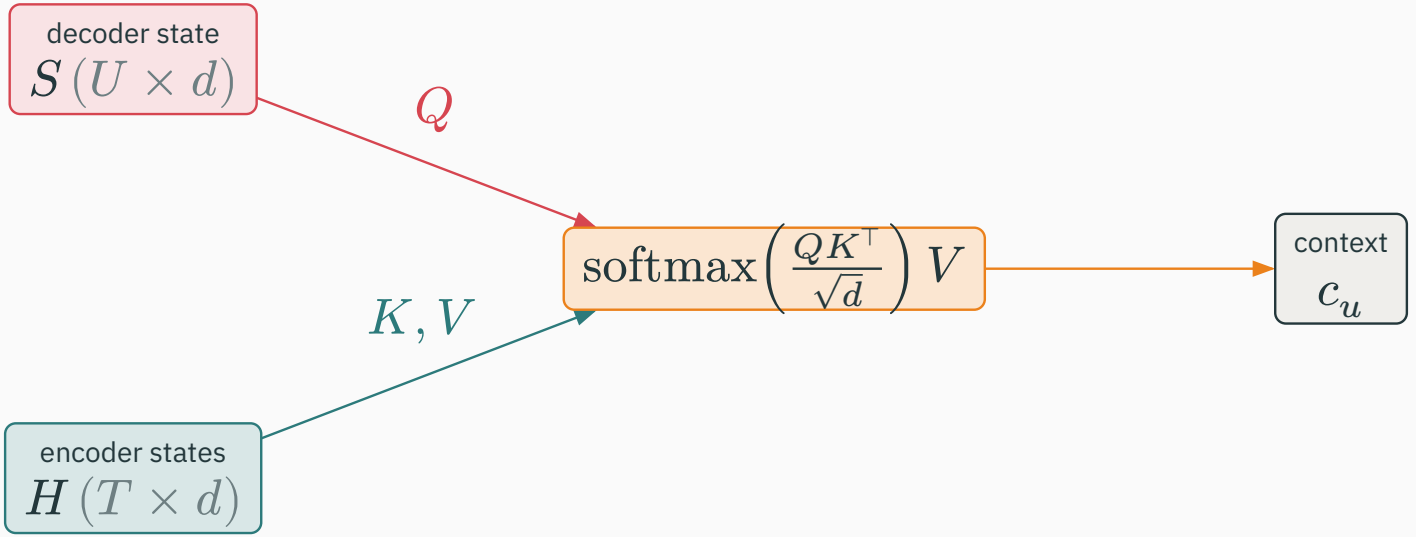
$$Q = SW_Q \ \text{(decoder)}, \quad K = HW_K, V = HW_V \ \text{(encoder)}$$

$$\text{CrossAttn}(S, H) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right) V$$

each target position **asks the source** which parts to attend to — this is how translation aligns

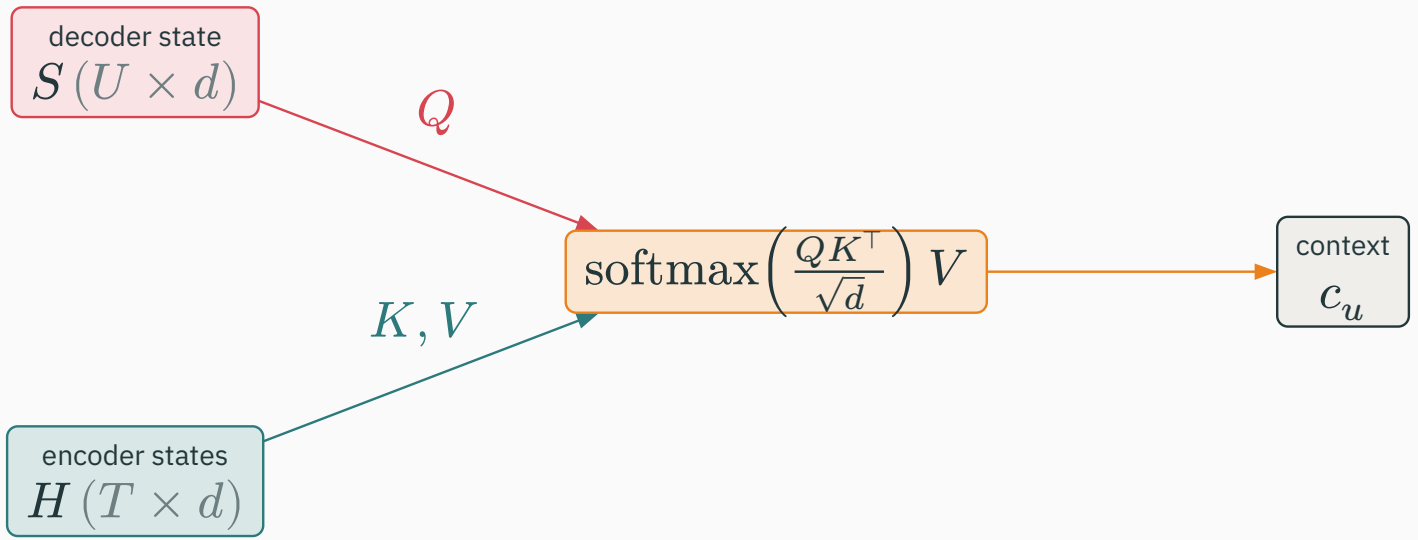
Cross-attention, drawn

V VISUAL



Cross-attention, drawn

V VISUAL



Q comes from the decoder state S; K and V come from the encoder states H — the decoder reads the source

Good for sequence-to-sequence

V VISUAL

Encoder–decoder fits tasks where output length $U \neq$ input length T:

Good for sequence-to-sequence

V VISUAL

Encoder–decoder fits tasks where output length $U \neq$ input length T:

Task	**Source → target**
translation	English sentence → German sentence
summarization	long document → short summary
question answering	question + context → answer
data-to-text	table / triples → sentence

Good for sequence-to-sequence

V VISUAL

Encoder–decoder fits tasks where output length $U \neq$ input length T:

Task	**Source → target**
translation	English sentence → German sentence
summarization	long document → short summary
question answering	question + context → answer
data-to-text	table / triples → sentence

a full re-encoding of the source, then free-length generation of the target

T5: text-to-text

D DERIVATION

Cast **every** task as text $\rightarrow$ text, so one model and one loss handle all of them:

T5: text-to-text

D DERIVATION

Cast **every** task as text → text, so one model and one loss handle all of them:

```
translate English to German: That is good.  ⟶  Das ist gut.
sst2 sentence: a dull film  ⟶  negative
```

T5: text-to-text

D DERIVATION

Cast **every** task as text → text, so one model and one loss handle all of them:

```
translate English to German: That is good.  ⟶  Das ist gut.
sst2 sentence: a dull film  ⟶  negative
```

Classification, translation, summarization — all become “read this text, write that text”. A task prefix tells the model what to do (Raffel et al.).

BART: denoising seq2seq

D DERIVATION

Corrupt the input, then reconstruct the **original** — a bidirectional encoder feeds a causal decoder:

BART: denoising seq2seq

D DERIVATION

Corrupt the input, then reconstruct the **original** — a bidirectional encoder feeds a causal decoder:

$$\tilde{x} = C(x) \xrightarrow{\text{Encoder–Decoder}} x$$

BART: denoising seq2seq

D DERIVATION

Corrupt the input, then reconstruct the **original** — a bidirectional encoder feeds a causal decoder:

$$\tilde{x} = C(x) \xrightarrow{\text{Encoder–Decoder}} x$$

Encoder — bidirectional, reads the corrupted text (like BERT).

Decoder — left-to-right, reconstructs the clean text (like GPT).

BART: denoising seq2seq

D DERIVATION

Corrupt the input, then reconstruct the **original** — a bidirectional encoder feeds a causal decoder:

$$\tilde{x} = C(x) \xrightarrow{\text{Encoder–Decoder}} x$$

Encoder — bidirectional, reads the corrupted text (like BERT).

Decoder — left-to-right, reconstructs the clean text (like GPT).

BART unifies BERT-style understanding and GPT-style generation in one seq2seq model (Lewis et al.)

Pretraining objectives compared

Three objectives

V VISUAL

Each family is defined by **what it must predict**:

Three objectives

V VISUAL

Each family is defined by **what it must predict**:

Objective	Input → predict	Architecture	Family
causal LM	prefix → next token	decoder-only	GPT
masked LM	corrupted → masked tokens	encoder-only	BERT
denoising seq2seq	corrupted → original	encoder–decoder	T5 · BART

Three objectives

V VISUAL

Each family is defined by **what it must predict**:

Objective	Input → predict	Architecture	Family
causal LM	prefix → next token	decoder-only	GPT
masked LM	corrupted → masked tokens	encoder-only	BERT
denoising seq2seq	corrupted → original	encoder–decoder	T5 · BART

same block; the **prediction target** is what differs

The three losses

D DERIVATION

The three losses

D DERIVATION

causal LM : $\mathcal{L}_{\text{CLM}} = -\sum_{t} \log p(x_t \mid x_{<t})$

The three losses

D DERIVATION

causal LM : $\mathcal{L}_{\text{CLM}} = -\sum_{t} \log p(x_t \mid x_{<t})$

masked LM : $\mathcal{L}_{\text{MLM}} = -\sum_{m \in M} \log p(x_m \mid \tilde{x})$

The three losses

D DERIVATION

causal LM : $\mathcal{L}_{\text{CLM}} = -\sum_{t} \log p(x_t \mid x_{<t})$

masked LM : $\mathcal{L}_{\text{MLM}} = -\sum_{m \in M} \log p(x_m \mid \tilde{x})$

denoising : $p(x \mid \tilde{x}) = \prod_{t} p(x_t \mid x_{<t}, \tilde{x}), \quad \tilde{x} = C(x)$

The three losses

D DERIVATION

causal LM : $\mathcal{L}_{\text{CLM}} = -\sum_{t} \log p(x_t \mid x_{<t})$

masked LM : $\mathcal{L}_{\text{MLM}} = -\sum_{m \in M} \log p(x_m \mid \tilde{x})$

denoising : $p(x \mid \tilde{x}) = \prod_{t} p(x_t \mid x_{<t}, \tilde{x}), \quad \tilde{x} = C(x)$

CLM $\longrightarrow$ generation · MLM $\longrightarrow$ understanding / bidirectional · denoising $\longrightarrow$ reconstruct target from input

Span corruption (T5-style)

D DERIVATION

Instead of masking single tokens, mask **contiguous spans** and replace each with one sentinel:

Span corruption (T5-style)

D DERIVATION

Instead of masking single tokens, mask **contiguous spans** and replace each with one sentinel:

input: `The quick brown fox jumps over the lazy dog`

corrupt: `The quick <X> jumps over <Y> dog`

target: `<X> brown fox <Y> the lazy`

Span corruption (T5-style)

D DERIVATION

Instead of masking single tokens, mask **contiguous spans** and replace each with one sentinel:

input: `The quick brown fox jumps over the lazy dog`

corrupt: `The quick <X> jumps over <Y> dog`

target: `<X> brown fox <Y> the lazy`

the decoder emits the **dropped spans**, tagged by sentinel — shorter targets, harder task

Which objective for which task

V VISUAL

You want	Pick	Why
classify / tag / retrieve	MLM (encoder)	both-side context
open-ended generation	causal LM (decoder)	native next-token
translate / summarize	seq2seq (enc–dec)	source $\rightarrow$ target
one model for everything	text-to-text (T5)	unified interface

Which objective for which task

V VISUAL

You want	Pick	Why
classify / tag / retrieve	MLM (encoder)	both-side context
open-ended generation	causal LM (decoder)	native next-token
translate / summarize	seq2seq (enc–dec)	source → target
one model for everything	text-to-text (T5)	unified interface

match the **objective** to the **shape** of your task

Finetuning and adaptation

Pretrain, then finetune

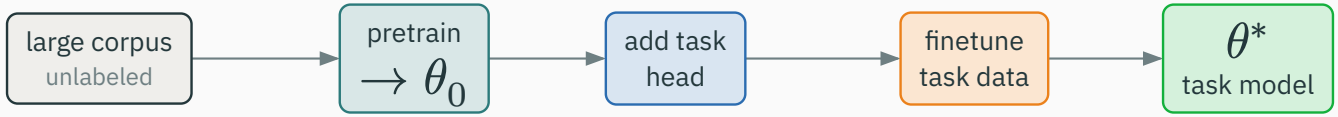
V VISUAL

The dominant paradigm: pretrain once on a huge corpus, then specialize cheaply:

Pretrain, then finetune

V VISUAL

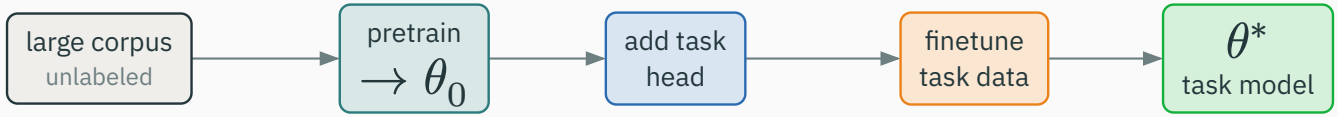
The dominant paradigm: pretrain once on a huge corpus, then specialize cheaply:



Pretrain, then finetune

V VISUAL

The dominant paradigm: pretrain once on a huge corpus, then specialize cheaply:

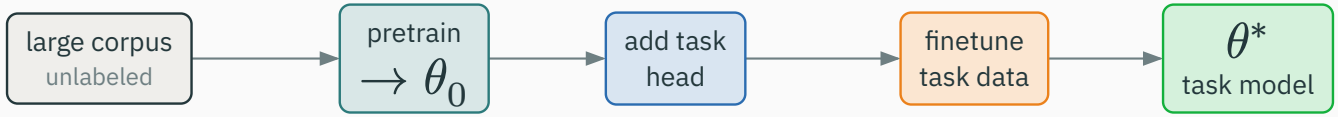


$$\theta_0 \text{ (pretrained)} \quad \rightarrow \quad \theta^* = \arg\min_{\theta} \mathcal{L}_{\text{task}(\theta)}$$

Pretrain, then finetune

V VISUAL

The dominant paradigm: pretrain once on a huge corpus, then specialize cheaply:



$$\theta_0 \text{ (pretrained)} \quad \rightarrow \quad \theta^* = \arg\min_{\theta} \mathcal{L}_{\text{task}(\theta)}$$

the pretrained weights θ_0 are the **starting point**; finetuning nudges them to the task

Full finetuning

D DERIVATION

Update **all** parameters on the task data:

Full finetuning

D DERIVATION

Update **all** parameters on the task data:

Pro — best adaptation; the whole network can reshape to the task.

Con — memory-heavy; can **overfit** small data; a **separate copy** of the model per task.

Full finetuning

D DERIVATION

Update **all** parameters on the task data:

Pro — best adaptation; the whole network can reshape to the task.

Con — memory-heavy; can **overfit** small data; a **separate copy** of the model per task.

the strong default when you have enough labelled data and compute

Feature extraction (frozen backbone)

D DERIVATION

Freeze the pretrained encoder; train **only** the task head:

Feature extraction (frozen backbone)

D DERIVATION

Freeze the pretrained encoder; train **only** the task head:

$$\underbrace{\theta_{\text{backbone}}}_{\text{frozen}} + \underbrace{\theta_{\text{head}}}_{\text{trained}}$$

Feature extraction (frozen backbone)

D DERIVATION

Freeze the pretrained encoder; train **only** the task head:

$$\underbrace{\theta_{\text{backbone}}}_{\text{frozen}} + \underbrace{\theta_{\text{head}}}_{\text{trained}}$$

- fast, cheap, tiny memory footprint;
- good for **small data** or a **quick baseline**;
- usually weaker than full finetuning.

Feature extraction (frozen backbone)

D DERIVATION

Freeze the pretrained encoder; train **only** the task head:

$$\underbrace{\theta_{\text{backbone}}}_{\text{frozen}} + \underbrace{\theta_{\text{head}}}_{\text{trained}}$$

- fast, cheap, tiny memory footprint;
- good for **small data** or a **quick baseline**;
- usually weaker than full finetuning.

the backbone is a fixed feature extractor; only a small head learns the task

PEFT preview

I INTERACTIVE

Between frozen and full: update a **tiny** set of extra parameters, keep the backbone fixed.

PEFT preview

I INTERACTIVE

Between frozen and full: update a **tiny** set of extra parameters, keep the backbone fixed.

Method	**Idea**
adapters	small bottleneck modules inserted per layer
prefix / prompt tuning	learn virtual tokens prepended to the input
LoRA	learn low-rank updates $\Delta W = BA$ to weights

PEFT preview

I INTERACTIVE

Between frozen and full: update a **tiny** set of extra parameters, keep the backbone fixed.

Method	Idea
adapters	small bottleneck modules inserted per layer
prefix / prompt tuning	learn virtual tokens prepended to the input
LoRA	learn low-rank updates $\Delta W = BA$ to weights

INTERACTIVE

↗ nipunbatra.github.io/interactive-articles/lora-adapter

Watch a low-rank update $\Delta W = BA$ adapt a frozen weight matrix — a full lecture on **parameter-efficient** finetuning comes later.

Classification-head example

D DERIVATION

Finetune BERT for sentiment — one small head over `[CLS]`:

Classification-head example

D DERIVATION

Finetune BERT for sentiment — one small head over `[CLS]`:

```
[CLS] the movie was wonderful [SEP]
```

Classification-head example

D DERIVATION

Finetune BERT for sentiment — one small head over `[CLS]`:

```
[CLS] the movie was wonderful [SEP]
```

$$z = W\,h_{[\mathrm{CLS}]} + b, \qquad \mathcal{L} = -\log \mathrm{softmax}(z)_{y^*}$$

Classification-head example

D DERIVATION

Finetune BERT for sentiment — one small head over `[CLS]`:

```
[CLS] the movie was wonderful [SEP]
```

$$z = W\, h_{[\mathrm{CLS}]} + b, \qquad \mathcal{L} = -\log \mathrm{softmax}(z)_{y^*}$$

pretrained encoder + a linear head + cross-entropy — the whole finetuning setup

Generation finetuning

D DERIVATION

For a generation task (e.g. summarization), finetune with the **seq2seq** loss:

Generation finetuning

D DERIVATION

For a generation task (e.g. summarization), finetune with the **seq2seq** loss:

$$\mathcal{L} = -\sum_{u=1}^{U} \log p_\theta(y_u \mid y_{<u}, x)$$

Generation finetuning

D DERIVATION

For a generation task (e.g. summarization), finetune with the **seq2seq** loss:

$$\mathcal{L} = -\sum_{u=1}^{U} \log p_\theta(y_u \mid y_{<u}, x)$$

- x is the document, y the reference summary;
- teacher forcing: feed the true $y_{<u}$ during training.

Generation finetuning

D DERIVATION

For a generation task (e.g. summarization), finetune with the **seq2seq** loss:

$$\mathcal{L} = -\sum_{u=1}^{U} \log p_\theta(y_u \mid y_{<u}, x)$$

- x is the document, y the reference summary;
- teacher forcing: feed the true $y_{<u}$ during training.

same next-token loss, now conditioned on the source x — the model learns to **transduce**

Decoding and generation

The decoding problem

Given the next-token distribution p_t, **how do we choose** the next token?

The decoding problem

Given the next-token distribution p_t, **how do we choose** the next token?

Training gives us $p_\theta(x_{t+1} \mid x_{\leq t})$. Decoding is a **separate** choice made at generation time — no weights change, but it hugely shapes the output.

The decoding problem

Given the next-token distribution p_t, **how do we choose** the next token?

Training gives us $p_\theta(x_{t+1} \mid x_{\leq t})$. Decoding is a **separate** choice made at generation time — no weights change, but it hugely shapes the output.

the same model can sound repetitive or creative depending only on the **decoding rule**

Greedy decoding

D DERIVATION

Always take the single most probable token:

Greedy decoding

D DERIVATION

Always take the single most probable token:

$$\hat{x}_{t+1} = \arg\max_{v} p_t(v)$$

Greedy decoding

D DERIVATION

Always take the single most probable token:

$$\hat{x}_{t+1} = \arg\max_{v} p_t(v)$$

Pro — simple, deterministic, fast; the model's local best guess.

Con — repetitive and bland; locally-best $\neq$ globally-best sequence.

Beam search

D DERIVATION

Keep the B highest-scoring **partial sequences** at each step, by cumulative log-probability:

Beam search

D DERIVATION

Keep the B highest-scoring **partial sequences** at each step, by cumulative log-probability:

$$\text{score}(y_{1:u}) = \sum_{k=1}^{u} \log p(y_k \mid y_{<k})$$

Beam search

D DERIVATION

Keep the B highest-scoring **partial sequences** at each step, by cumulative log-probability:

$$\text{score}(y_{1:u}) = \sum_{k=1}^{u} \log p(y_k \mid y_{<k})$$

- explores several hypotheses in parallel, not just one;
- **good** for translation / summarization (a single high-likelihood answer);
- **poor** for open-ended text — tends to be generic and repetitive.

Beam search

D DERIVATION

Keep the B highest-scoring **partial sequences** at each step, by cumulative log-probability:

$$\text{score}(y_{1:u}) = \sum_{k=1}^{u} \log p(y_k \mid y_{<k})$$

- explores several hypotheses in parallel, not just one;
- **good** for translation / summarization (a single high-likelihood answer);
- **poor** for open-ended text — tends to be generic and repetitive.

$B = 1$ is greedy; larger B searches wider but costs more

Temperature

D DERIVATION

Rescale the logits by τ **before** the softmax to control sharpness:

Temperature

D DERIVATION

Rescale the logits by τ **before** the softmax to control sharpness:

$$p_i(\tau) = \mathrm{softmax}(z/\tau)_i$$

Temperature

D DERIVATION

Rescale the logits by τ **before** the softmax to control sharpness:

$$p_i(\tau) = \mathrm{softmax}(z/\tau)_i$$

τ	**Effect**
$\tau < 1$	sharper — bolder, more deterministic
$\tau = 1$	the model's own distribution
$\tau > 1$	flatter — more random, more diverse

Temperature

D DERIVATION

Rescale the logits by τ **before** the softmax to control sharpness:

$$p_i(\tau) = \mathrm{softmax}(z/\tau)_i$$

τ	**Effect**
$\tau < 1$	sharper — bolder, more deterministic
$\tau = 1$	the model's own distribution
$\tau > 1$	flatter — more random, more diverse

$\tau \to 0$ recovers greedy; $\tau \to \infty$ approaches uniform

Top-k and nucleus (top-p)

D DERIVATION

Truncate the tail **before** sampling, so rare tokens can’t derail generation:

Top-k and nucleus (top-p)

D DERIVATION

Truncate the tail **before** sampling, so rare tokens can't derail generation:

Top-k — keep the k most probable tokens, renormalize, sample.

Nucleus (top-p) — keep the smallest set S with $\sum_{i \in S} p_i \geq p$, then sample.

Top-k and nucleus (top-p)

D DERIVATION

Truncate the tail **before** sampling, so rare tokens can't derail generation:

Top-k — keep the k most probable tokens, renormalize, sample.

Nucleus (top-p) — keep the smallest set S with $\sum_{i \in S} p_i \geq p$, then sample.

top-p **adapts** the candidate set to the model's confidence — narrow when sure, wide when unsure

Repetition penalties

D DERIVATION

Explicit fixes for the degenerate repetition that sampling and greedy can produce:

Repetition penalties

D DERIVATION

Explicit fixes for the degenerate repetition that sampling and greedy can produce:

- **no-repeat n-gram** — forbid emitting an n-gram already seen;
- **repetition / frequency penalty** — down-weight tokens already generated;
- **presence penalty** — discourage reusing any token that has appeared.

Repetition penalties

D DERIVATION

Explicit fixes for the degenerate repetition that sampling and greedy can produce:

- **no-repeat n-gram** — forbid emitting an n-gram already seen;
- **repetition / frequency penalty** — down-weight tokens already generated;
- **presence penalty** — discourage reusing any token that has appeared.

cheap, decoding-time nudges away from loops — no retraining

Interactive: the decoding lab

I INTERACTIVE

INTERACTIVE

↗ nipunbatra.github.io/interactive-articles/softmax-temperature

Sweep **temperature**, **top-k** and **top-p** on a fixed set of logits and watch the next-token bar chart sharpen or flatten — then sample repeatedly and see the diversity of outputs change.

Worked examples

BERT MLM: the loss

D DERIVATION

Sentence `I like [MASK]`; candidates and BERT's predicted probabilities:

BERT MLM: the loss

D DERIVATION

Sentence `I like [MASK]`; candidates and BERT's predicted probabilities:

token	`cats`	`blue`	`running`
p	**0.80**	0.05	0.15

BERT MLM: the loss

D DERIVATION

Sentence `I like [MASK]`; candidates and BERT's predicted probabilities:

token	`cats`	`blue`	`running`
p	**0.80**	0.05	0.15

Target is `cats`; cross-entropy picks out its probability:

$$\mathcal{L} = -\log(0.80) \approx 0.223$$

BERT MLM: the loss

D DERIVATION

Sentence `I like [MASK]`; candidates and BERT's predicted probabilities:

token	`cats`	`blue`	`running`
p	**0.80**	0.05	0.15

Target is `cats`; cross-entropy picks out its probability:

$$\mathcal{L} = -\log(0.80) \approx 0.223$$

a confident, correct prediction — small loss; a perfect one would give 0

GPT next-token: the loss

D DERIVATION

Prompt `I like`; the decoder’s next-token distribution:

GPT next-token: the loss

D DERIVATION

Prompt `I like`; the decoder's next-token distribution:

token	`cats`	`dogs`	`blue`
p	**0.6**	0.3	0.1

GPT next-token: the loss

D DERIVATION

Prompt `I like`; the decoder's next-token distribution:

token	`cats`	`dogs`	`blue`
p	**0.6**	0.3	0.1

Target is `cats`:

$$\mathcal{L} = -\log(0.6) \approx 0.511$$

GPT next-token: the loss

D DERIVATION

Prompt `I like`; the decoder's next-token distribution:

token	`cats`	`dogs`	`blue`
p	**0.6**	0.3	0.1

Target is `cats`:

$$\mathcal{L} = -\log(0.6) \approx 0.511$$

same cross-entropy loss as MLM — the difference is **causal** context, not left-and-right

T5 text-to-text: the loss

D DERIVATION

Prompt the encoder, generate the target with the decoder:

T5 text-to-text: the loss

D DERIVATION

Prompt the encoder, generate the target with the decoder:

`translate English to Hindi: I like cats` $\longrightarrow$ target sequence

T5 text-to-text: the loss

D DERIVATION

Prompt the encoder, generate the target with the decoder:

`translate English to Hindi: I like cats` $\longrightarrow$ target sequence

Sum the per-token negative log-likelihood over the target, conditioned on the source:

$$\mathcal{L} = -\sum_{u=1}^{U} \log p_\theta(y_u \mid y_{<u}, x)$$

T5 text-to-text: the loss

D DERIVATION

Prompt the encoder, generate the target with the decoder:

`translate English to Hindi: I like cats` $\longrightarrow$ target sequence

Sum the per-token negative log-likelihood over the target, conditioned on the source:

$$\mathcal{L} = -\sum_{u=1}^{U} \log p_\theta(y_u \mid y_{<u}, x)$$

exactly the seq2seq loss — the **source** x conditions every target-token prediction

Choosing a model in practice

Classification

D DERIVATION

Three valid routes — pick by data and infrastructure:

Classification

D DERIVATION

Three valid routes — pick by data and infrastructure:

Approach	How
encoder-only	`[CLS]` → classifier head
decoder-only	prompt → score label tokens
encoder–decoder	input → generate label text

Classification

D DERIVATION

Three valid routes — pick by data and infrastructure:

Approach	How
encoder-only	`[CLS]` → classifier head
decoder-only	prompt → score label tokens
encoder–decoder	input → generate label text

encoder-only is the classic strong, cheap default for a fixed label set

Generation

Generation

D DERIVATION

Need	Choice
open-ended text	decoder-only, sampling (top-p)
input-conditioned output	encoder–decoder
translation	beam search (high-likelihood answer)
creative / diverse	sampling with temperature

Generation

D DERIVATION

Need	Choice
open-ended text	decoder-only, sampling (top-p)
input-conditioned output	encoder–decoder
translation	beam search (high-likelihood answer)
creative / diverse	sampling with temperature

decode **deterministically** when there's one right answer; **sample** when diversity matters

Token tagging and small data

D DERIVATION

Token tagging — encoder-only. Both-side context helps label each token: $h_t \to y_t$ (NER, POS).

Small data — start from a **pretrained** model; escalate frozen $\to$ PEFT $\to$ full only as needed.

Token tagging and small data

D DERIVATION

Token tagging — encoder-only. Both-side context helps label each token: $h_t \to y_t$ (NER, POS).

Small data — start from a **pretrained** model; escalate frozen → PEFT → full only as needed.

always start pretrained; keep a held-out validation set and watch for overfitting

Limitations

Context length and cost

D DERIVATION

Self-attention is **quadratic** in sequence length, and context is **finite**:

Context length and cost

D DERIVATION

Self-attention is **quadratic** in sequence length, and context is **finite**:

$$O(T^2 d) \text{ compute}, \qquad T \leq T_{\max}$$

Context length and cost

D DERIVATION

Self-attention is **quadratic** in sequence length, and context is **finite**:

$$O(T^2 d) \text{ compute}, \qquad T \le T_{\max}$$

Every token attends every token, so cost grows with T^2. Models have a hard context limit $T_{\max}$; long inputs must be truncated or chunked.

Context length and cost

D DERIVATION

Self-attention is **quadratic** in sequence length, and context is **finite**:

$O(T^2 d)$ compute, $\qquad T \le T_{\max}$

Every token attends every token, so cost grows with T^2. Models have a hard context limit $T_{\max}$; long inputs must be truncated or chunked.

efficient / long-context attention is a whole topic for a later lecture

Tokenization issues

D DERIVATION

The tokenizer silently shapes everything downstream:

Tokenization issues

D DERIVATION

The tokenizer silently shapes everything downstream:

- text is split into **subwords** (BPE) — not words or characters;
- **script differences** change how many tokens a language needs;
- sequence length in tokens depends on the tokenizer, not just the text.

Tokenization issues

D DERIVATION

The tokenizer silently shapes everything downstream:

- text is split into **subwords** (BPE) — not words or characters;
- **script differences** change how many tokens a language needs;
- sequence length in tokens depends on the tokenizer, not just the text.

Perplexity is not comparable across tokenizers — a character model and a subword model predict different-sized units. Never compare PPL across vocabularies.

Worked BPE: learn one subword

D DERIVATION

Start from characters. In a toy corpus with `low` three times and `lower` once:

Worked BPE: learn one subword

D DERIVATION

Start from characters. In a toy corpus with `low` three times and `lower` once:

word	initial symbols
`low` × 3	`l o w </w>`
`lower` × 1	`l o w e r </w>`

Worked BPE: learn one subword

D DERIVATION

Start from characters. In a toy corpus with `low` three times and `lower` once:

word	initial symbols
`low` × 3	`l o w </w>`
`lower` × 1	`l o w e r </w>`

The adjacent pairs `(l, o)` and `(o, w)` each occur four times. Choose one tie arbitrarily: $lo \to \mathrm{lo}$, then the next frequent pair can give $\mathrm{lo}\ w \to \mathrm{low}$.

Worked BPE: learn one subword

D DERIVATION

Start from characters. In a toy corpus with `low` three times and `lower` once:

word	initial symbols
`low` × 3	`l o w </w>`
`lower` × 1	`l o w e r </w>`

The adjacent pairs `(l, o)` and `(o, w)` each occur four times. Choose one tie arbitrarily: $lo \to \mathrm{lo}$, then the next frequent pair can give $\mathrm{lo}\ w \to \mathrm{low}$.

after those two merges, an unseen `lowest` can begin as `[low, e, s, t, </w>]` — frequent pieces are reused.

Worked BPE: learn one subword

D DERIVATION

Start from characters. In a toy corpus with `low` three times and `lower` once:

word	initial symbols
`low` × 3	`l o w </w>`
`lower` × 1	`l o w e r </w>`

The adjacent pairs `(l, o)` and `(o, w)` each occur four times. Choose one tie arbitrarily: $lo \rightarrow \mathrm{lo}$, then the next frequent pair can give $\mathrm{lo}\ w \rightarrow \mathrm{low}$.

after those two merges, an unseen `lowest` can begin as `[low, e, s, t, </w>]` — frequent pieces are reused.

BPE discovers frequent character sequences, not guaranteed linguistic morphemes.

Interactive: tokenization

I INTERACTIVE

INTERACTIVE

↗ nipunbatra.github.io/interactive-articles/bpe-merges

Watch **byte-pair encoding** merge frequent character pairs into subword tokens — and see how the same sentence becomes a different number of tokens under different vocabularies.

Pretraining data matters

D DERIVATION

A pretrained model inherits its **corpus**:

Pretraining data matters

D DERIVATION

A pretrained model inherits its **corpus**:

- **biases** and toxicity present in the data;
- **coverage** — domains, languages, styles it did or didn't see;
- **factual staleness** — knowledge frozen at the training cutoff.

Pretraining data matters

D DERIVATION

A pretrained model inherits its **corpus**:

- **biases** and toxicity present in the data;
- **coverage** — domains, languages, styles it did or didn't see;
- **factual staleness** — knowledge frozen at the training cutoff.

the model is a mirror of its data — know what it was trained on before you trust it

Evaluation depends on the task

V VISUAL

No single number fits every model family:

Evaluation depends on the task

V VISUAL

No single number fits every model family:

Task type	Metrics
classification	accuracy · F1 · AUROC
generation	BLEU · ROUGE · BERTScore · human
language modeling	NLL · perplexity
retrieval	MRR · Recall@k · NDCG

Evaluation depends on the task

V VISUAL

No single number fits every model family:

Task type	Metrics
classification	accuracy · F1 · AUROC
generation	BLEU · ROUGE · BERTScore · human
language modeling	NLL · perplexity
retrieval	MRR · Recall@k · NDCG

choose the metric that matches the **task shape**, not the architecture

Summary

The model families, one table

V VISUAL

	Encoder-only	Decoder-only	Encoder–decoder
mask	bidirectional	causal	bidir + causal
pretraining	masked LM	next-token	denoise / seq2seq
best fit	understanding	generation	transduction
example	BERT · RoBERTa	GPT	T5 · BART

The model families, one table

V VISUAL

	Encoder-only	Decoder-only	Encoder–decoder
mask	bidirectional	causal	bidir + causal
pretraining	masked LM	next-token	denoise / seq2seq
best fit	understanding	generation	transduction
example	BERT · RoBERTa	GPT	T5 · BART

mask × pretraining objective × head ⇒ the family and its sweet spot

The objectives, in one breath

D DERIVATION

The objectives, in one breath

D DERIVATION

- **BERT** — predict the **missing tokens** using **both sides** (understanding);

The objectives, in one breath

D DERIVATION

- **BERT** — predict the **missing tokens** using **both sides** (understanding);
- **GPT** — predict the **next token** from the **previous** ones (generation);

The objectives, in one breath

D DERIVATION

- **BERT** — predict the **missing tokens** using **both sides** (understanding);
- **GPT** — predict the **next token** from the **previous** ones (generation);
- **T5 / BART** — **reconstruct or generate** the target from a corrupted / source input (transduction).

The objectives, in one breath

D DERIVATION

- **BERT** — predict the **missing tokens** using **both sides** (understanding);
- **GPT** — predict the **next token** from the **previous** ones (generation);
- **T5 / BART** — **reconstruct or generate** the target from a corrupted / source input (transduction).

one Transformer block; three ways to train it, three things it learns to do

Final mental model — three knobs

Architecture controls **information flow** — who attends to whom.

Objective controls **what it learns to do** — predict, mask, or reconstruct.

Finetuning / prompting controls **task adaptation** — how we specialize it.

$$\text{mask} + \text{objective} + \text{head} \Rightarrow \text{model family}$$

Same block — three families, by **mask**, **objective** and **head**.

Next: **Vision Transformers & multimodal** — images as patches, ViT, and CLIP.