Deep Generative Models

Chapter 1: Text Generation via Language Models

Ali Bereyhi

ali.bereyhi@utoronto.ca

Department of Electrical and Computer Engineering

University of Toronto

Summer 2025

LLMs: *Scale*

Since the introduction of transformer-based LMs, *various companies and research teams have trained these models over very large datasets*

Large LMs

They are called LLMs, as they are rather deep and trained on very large data

+ *How large are these models?*
– As time passes, they get deeper and wider

LLMs: *Example of GPT*

For instance the generative pre-trained model (GPT) has evolved as

1. *GPT-1 had* $L = 12$ *layers of multi-head SA* $\rightsquigarrow \approx$ *117 million parameters*
2. *GPT-2 got* $L = 48$ *layers* $\rightsquigarrow \approx$ *1.5 billions parameters*
3. *GPT-3 got* $L = 96$ *layers* $\rightsquigarrow \approx$ *175 billions parameters*
4. *GPT-4* $\rightsquigarrow$?

The larger they got, the more concerns raised $\rightsquigarrow$ *publicized details got reduced*

	Model	Data	Code
GPT-1	✔	✔	✔
GPT-2	✔	✔	✔
GPT-3	✔	⚠	✖
GPT-4	✖	✖	✖

Data for LLMs

+ *How large is data then? Where do they get it?*
– Well! Crazy large data collected from any possible text resources

---

The most common resources to collect data for training LLMs are

- ***Web-Text*** *which contains high quality texts from internet*
  ↳ *Sources like Reddit and Quora*
- ***Wikipedia*** *articles*
- ***Coding-Corpora*** *collected from various online resources*
  ↳ *Sources like GitHub and StackOverflow*
- ***Book-Corpora*** *which is made by including texts from reliable books*
- ***Common Crawl*** *that is a service providing web crawling*
  ↳ *Crawls need to be filtered and cleaned*
  ↳ *They need to be filtered due to safety and ethical issues*

Data: *Example – GPT*

For the example of GPT, we do know the data scale to some extent

1. *GPT-1 was trained on BookCorpus $\rightsquigarrow \approx$ 1 billion words*
2. *GPT-2 was trained on WebText created by OpenAI $\rightsquigarrow \approx$ 40 GB text*
3. *GPT-3 was trained on a giant dataset $\rightsquigarrow$ based on GPT-3 Paper*

Data	**Size in # Tokens**	**Weight**	**Eff. Epochs**
Common Crawl	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

4. *GPT-4 $\rightsquigarrow$?*

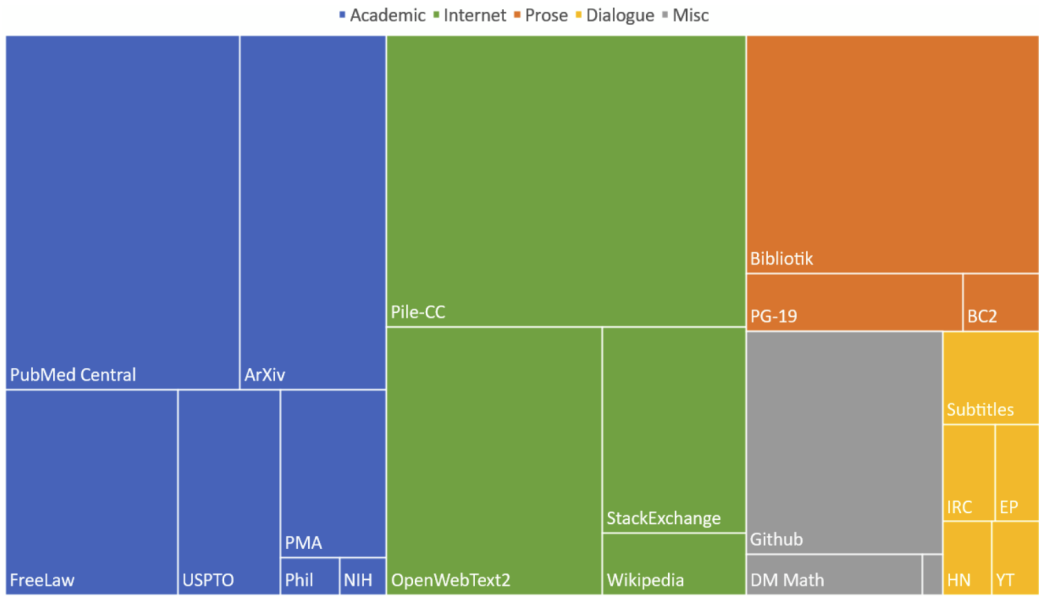
Data: *Practical Aspects*

Getting access to fair data is crucial in training LLMs

- *Several projects started to develop datasets*
  - ↳ *Some are public, some partially shared, and some have been kept private*

The Pile

The Pile is a large public dataset for LLM training



Data: *Ethical Challenges*

Attention

Data collection for LLM training faces various legal and ethical challenges

- ! *We should take care of individuals' privacy*
- ! *We must filter raw data to ensure training on ethically sound content*
- ? *How can we make sure about safety aspects of trained model*
- ! *We should make sure there is no copyright violations*
- ? *How can we guarantee rights of intellectual properties*

And believe it or not . . .

there are examples for each of these aspects[1]

[1] Just take a look at further reads on course page

Evaluation: *Perplexity*

Perplexity

Let a LM with weights $\mathbf{w}$ completes token sequence $x_1, x_2, \ldots, x_t$ with L tokens as $x_{t+1}, \ldots, x_{t+L}$. The perplexity is then defined as

$$\mathrm{PPL}_{\mathbf{w}}(x_{:}) = \left[P_{\mathbf{w}}\left(x_{t+1:t+L} | x_{1:t}\right)\right]^{-1/L}$$

Note that *perplexity* is reversely proportional to *likelihood*

$$\begin{aligned}
\mathrm{PPL}_{\mathbf{w}}(x_{:}) &= \sqrt[L]{\frac{1}{P_{\mathbf{w}}\left(x_{t+1:t+L} | x_{1:t}\right)}} \\
&= \sqrt[L]{\frac{1}{P_{\mathbf{w}}\left(x_{t+1} | x_{1:t}\right) P_{\mathbf{w}}\left(x_{t+2} | x_{1:t+1}\right) \ldots P_{\mathbf{w}}\left(x_{t+L} | x_{1:t+L-1}\right)}}
\end{aligned}$$

Evaluation: *Perplexity*

In practice, we often work with *log-perplexity*, *i.e.*,

$$\log \mathrm{PPL}_{\mathbf{w}}(x_{:}) = -\frac{1}{L} \log P_{\mathbf{w}}(x_{t+1:t+L} | x_{1:t})$$
$$= -\frac{1}{L} \sum_{\ell=1}^{L} \log P_{\mathbf{w}}(x_{t+\ell} | x_{1:t+\ell-1})$$

which is literally negative average log-likelihood

Properties of log-Perplexity

It is easy to see that

1. *log-Perplexity is always non-negative*
2. *smaller log-Perplexity* $\rightsquigarrow$ *larger likelihood* $\rightsquigarrow$ *more-reliable LM*

Back to Technical Aspects

+ *So, you mean that by training these giant models for a long time, they will start to do what ChatGPT does?*
– Well, not immediately! We need a few further steps

The trained LLMs are indeed only text completer

> How can I get rid of Ali Bereyhi ⇜ $x_{1:t}$
>
> ... who works at the ECE department in UofT and is kind of annoying, especially in the Applied Deep Learning course where he talks about backpropagation for ever! ⇜ $x_{t+1:T}$

This is why we call these models pre-trained models

Pre-training

Pre-training

The process of training a LM on a large language dataset is called pre-training

A pre-trained LM is able to sample from the *generic language distribution*

$$p\left(x_{t+1:T} | x_{1:t}\right)$$

But, we often want them to sample from a *task-specific distribution*

$$p\left(y_{1:L} | x_{1:t}\right)$$

Example: *To learn to give answer to a question*

How can I get rid of Ali Bereyhi	$\rightsquigarrow x_{1:t}$
Just ask questions!	$\rightsquigarrow y_{1:L}$

Fine-Tuning a Pre-trained Model

In general, task-specific distributions are *close* to what LM learned

- ✔ *LM can generate meaningful and syntactically-correct completions*
  - ↳ *There all lots of such completions*
  - ↳ *A few address the one required in a given task like Question Answering*
- ✖ *After pre-training it has no priority for certain completions*
  - ↳ *As it does not know which one is the desired one*

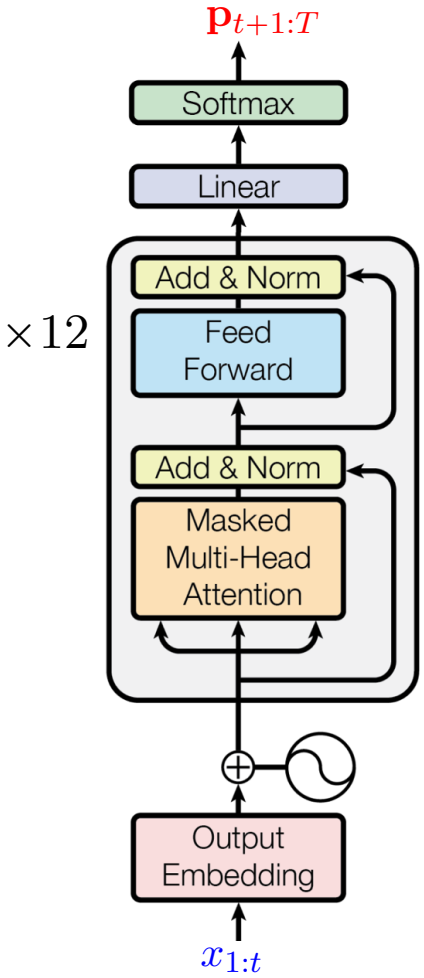
We can hence adapt the pre-trained LM to task-specific distribution

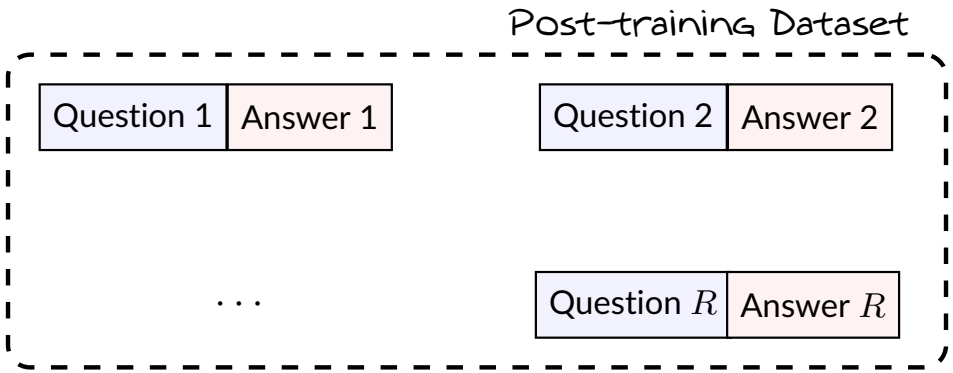
via fine-tuning them for the task

Fine-Tuning

Further training of pre-trained LMs on a small dataset of a specific task

Fine-Tuning *GPT-1 for Q&A*





Post-training Samples:

<STR> starting tokens

<delimiter> delimiter tokens indicating end of question

<END> end-of-sample tokens

Fine-Tuning *GPT-1 for Q&A*

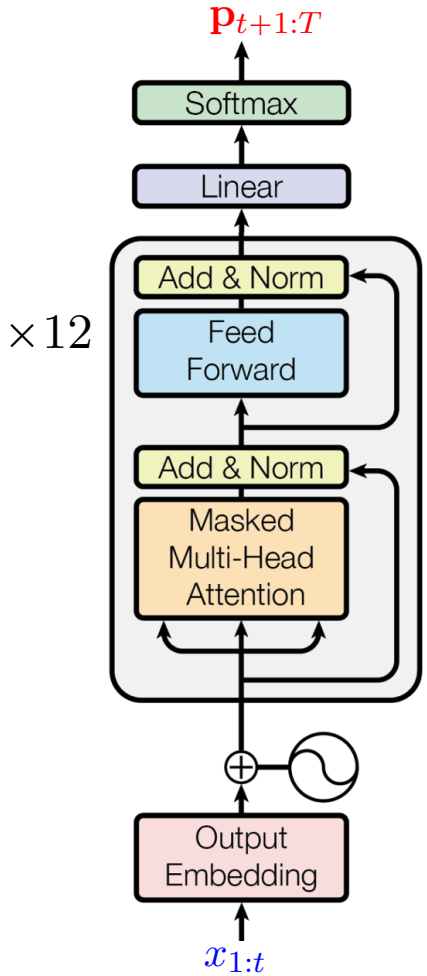
Staring from the pre-trained GPT-1, we train the model for a few epochs on thepost-training dataset made out of sample questions and answers

- *Model is trained as usual on structured samples*
  - ↳ *Since we label samples, we call it supervised fine-thing*
- *Post-training data size and duration are significantly less than pre-training*
  - ↳ *For Question Answering, GPT-1 used RACE dataset with 100K questions*
  - ↳ *GPT-1 was fine-tuned for only 3 epochs*

Note that fine-tuning can be in general done for any task

- *Text classification*
- *Similarity check*
- *Question answering*
- . . .

Sampling *Answers* from Fine-Tuned *GPT-1*



Question ⋆	??

Set $x_{1:t}$ to

<STR>	Question ⋆	<delimiter>

and let the fine-tuned model completes

Supervised Fine-Tuning LLMs: *Formulation*

We can fit fine-tuning in our generic formulation of language modeling: *to this end, let us specify the main two notions we deal with*

Pre-trained Distribution

Through pre-training LLM learns to approximate a generic distribution as

$$P_{\mathbf{w}}\left(x_{t+1:T}|x_{1:t}\right)$$

with some model that is parameterized by $\mathbf{w}$

Task-specific Distribution

A given language task is described by its task-specific distribution

$$Q\left(y_{1:N}|u_{1:M}\right)$$

with $u_{1:M}$ *being the task prompt and* $y_{1:N}$ *is the coherent response*

Supervised Fine-Tuning LLMs: *Formulation*

A supervised fine-tuning algorithm works as follows: *it takes two inputs*

1. *Pre-trained LLM with distribution* $P_{\mathbf{w}}\left(x_{t+1:T}|x_{1:t}\right)$
2. *Samples of task-specific distribution* $\{(y_{:,i}, u_{:,i}) \quad \text{for} \quad i = 1:R\}$

and performs the following step

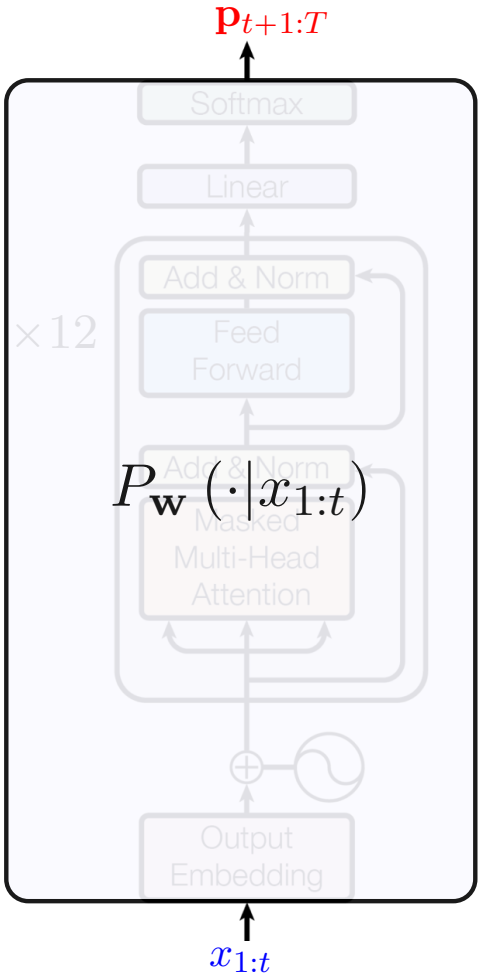
1. *It makes a dataset by structuring task-specific samples*

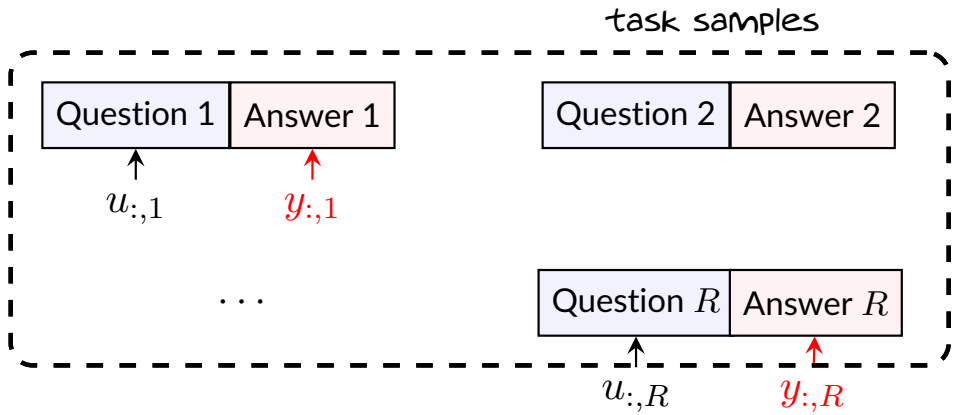
$$\mathbb{Q} = \{\{x_{1:T}\}_i \quad \text{for} \quad i = 1:R\} : \{x_{1:T}\}_i \leftarrow F\left(y_{:,i}, u_{:,i}\right)$$

   ↳ $F\left(y_{:,i}, u_{:,i}\right)$ *is a simple structuring function*
   ↳ *It describes how to get* $x_{1:T}$ *from* $(y_{:}, u_{:})$ *and vice versa*

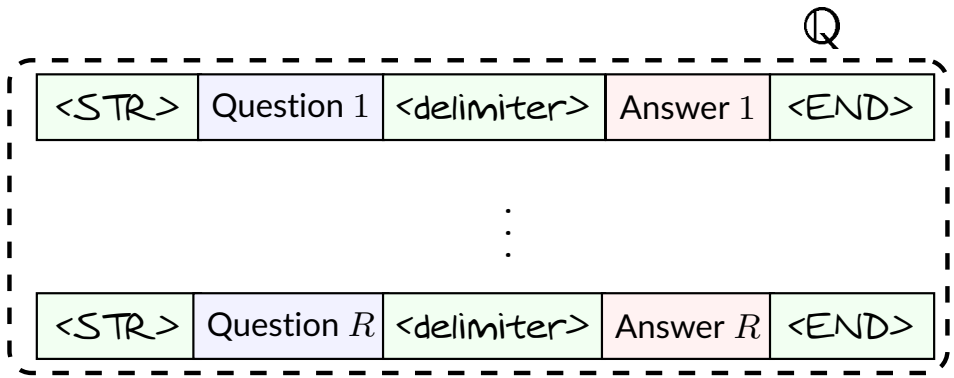
2. *It starts with* $\mathbf{w}$ *and trains LLM on* $\mathbb{Q}$ *for further epochs*

Supervised Fine-Tuning LLMs: *Q&A Example*





$F(y_{:,r}, u_{:,r})$:



Sampling Fine-Tuned LLMs: *Formulation*

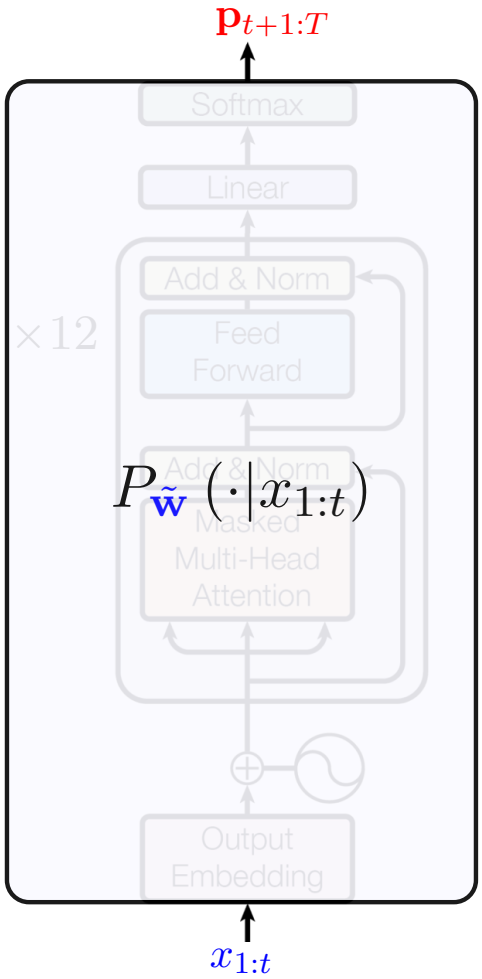
After fine-tuning, *the LLM is updated to* $P_{\tilde{\mathbf{w}}}\left(x_{t+1:T}|x_{1:t}\right)$*: to sample from it*

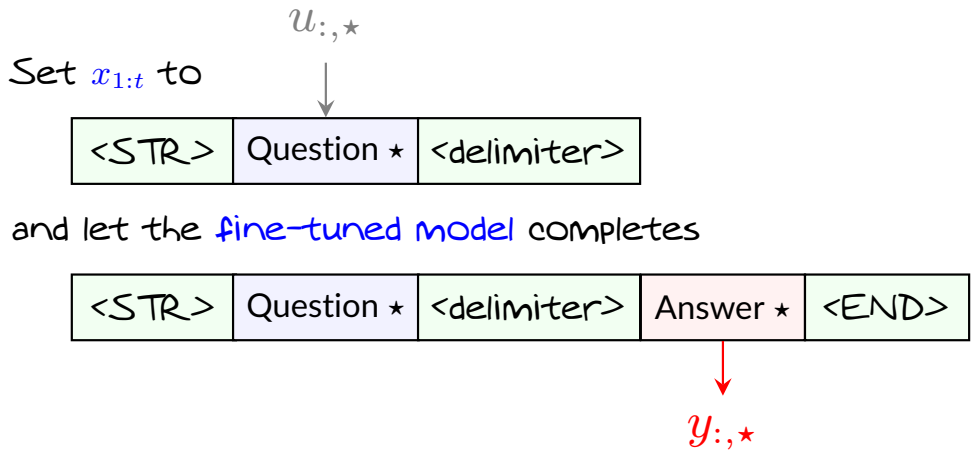
1. *Make an input token as*

$$x_{1:t} \propto F\left(\varnothing, u_{:}\right)$$

2. *Sample* $x_{t+1:T}$ *from the fine-tuned model*
3. *Use* F *to get back to* $y_{:}$

Sampling Fine-Tuned LLMs: *Q&A Example*





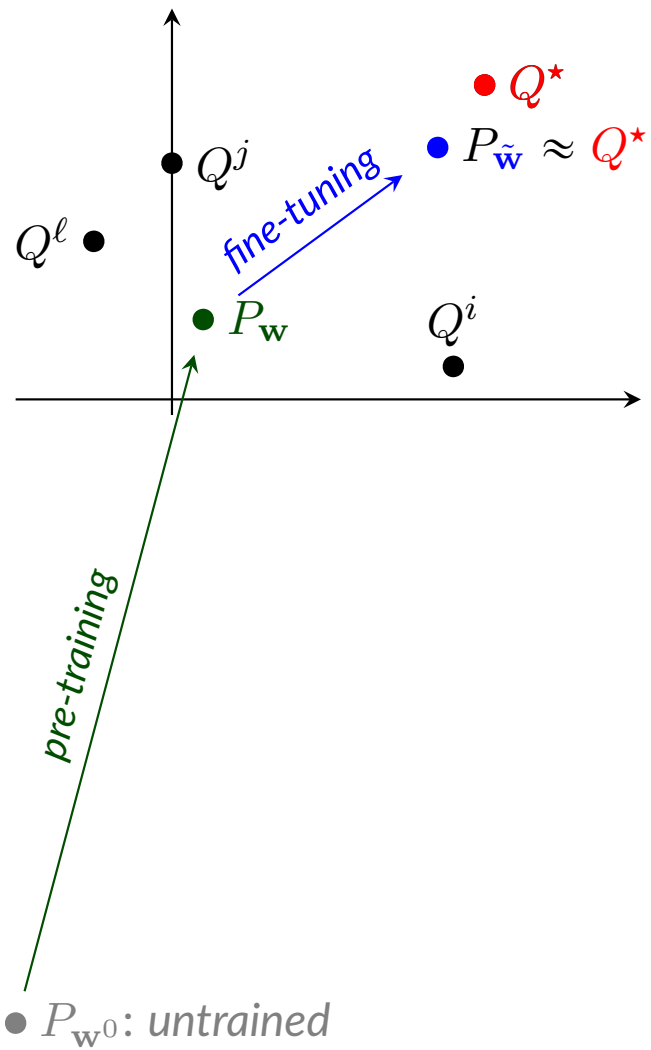
Fine-Tuning ≡ *Slight Change in Learned Distribution*

Intuitively, pre-training trains LM to

- *represent generic-task distribution*
- *get close to any language task*

fine-tuning focuses on a task pushing LM to

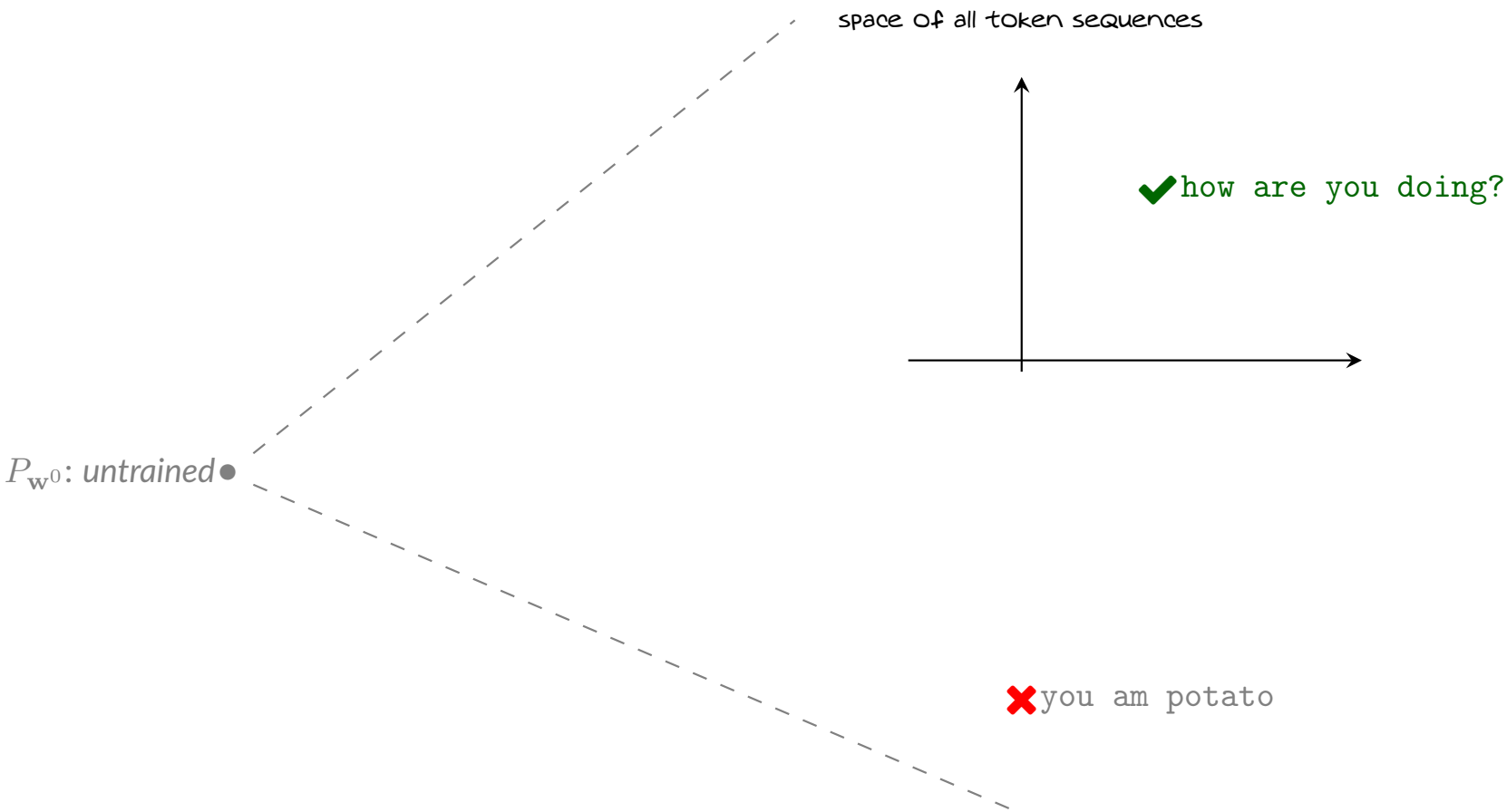
- *represent the task-specific distribution*
- *could get far from other tasks*



Fine-Tuning ≡ *Limiting Sampling Space*

Alternative Intuition

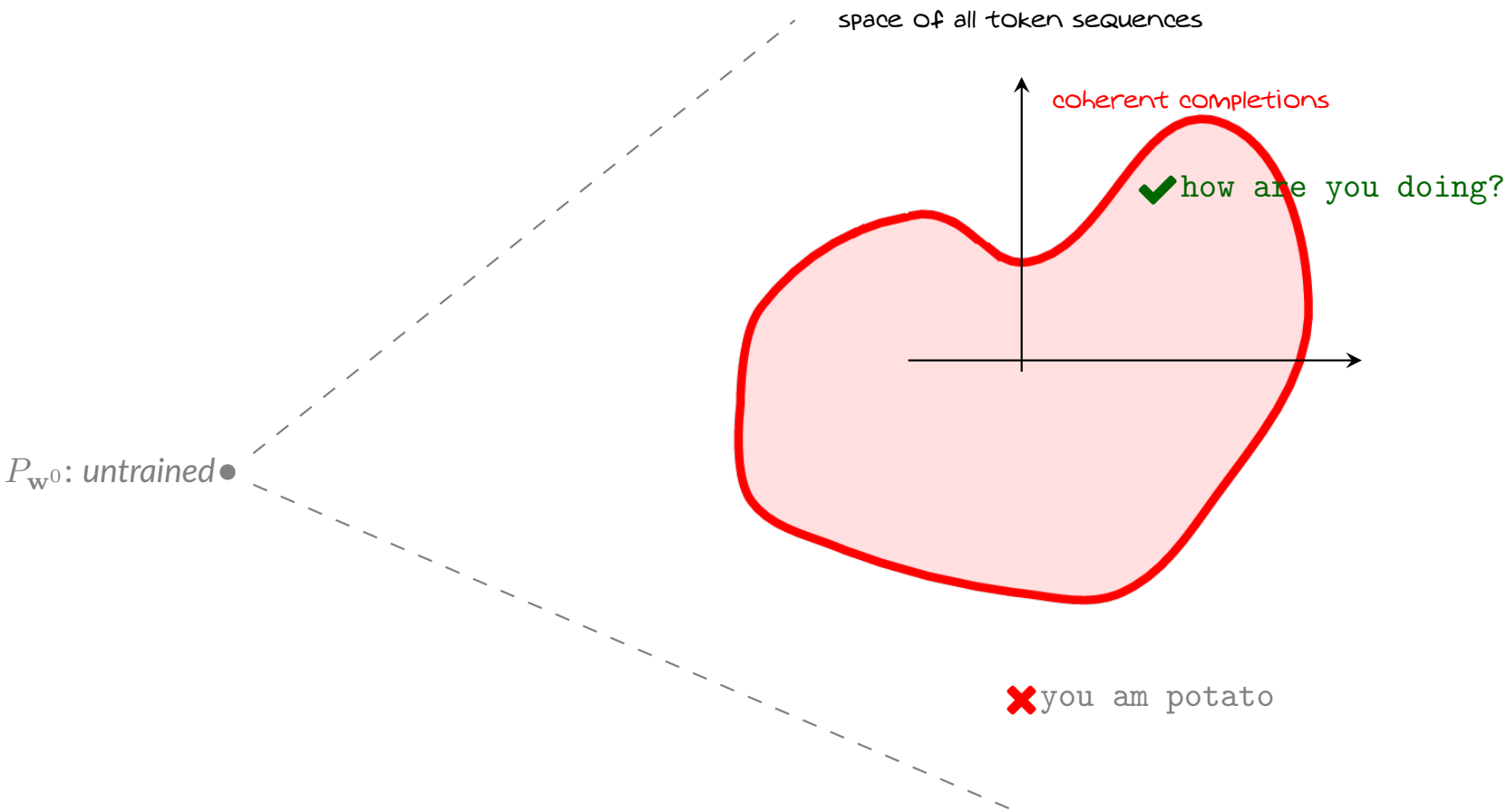
Pre-training trains LM to give coherent completions while fine-tuning restricts the completion to be the one desired for the task



Fine-Tuning $\equiv$ *Limiting Sampling Space*

Alternative Intuition

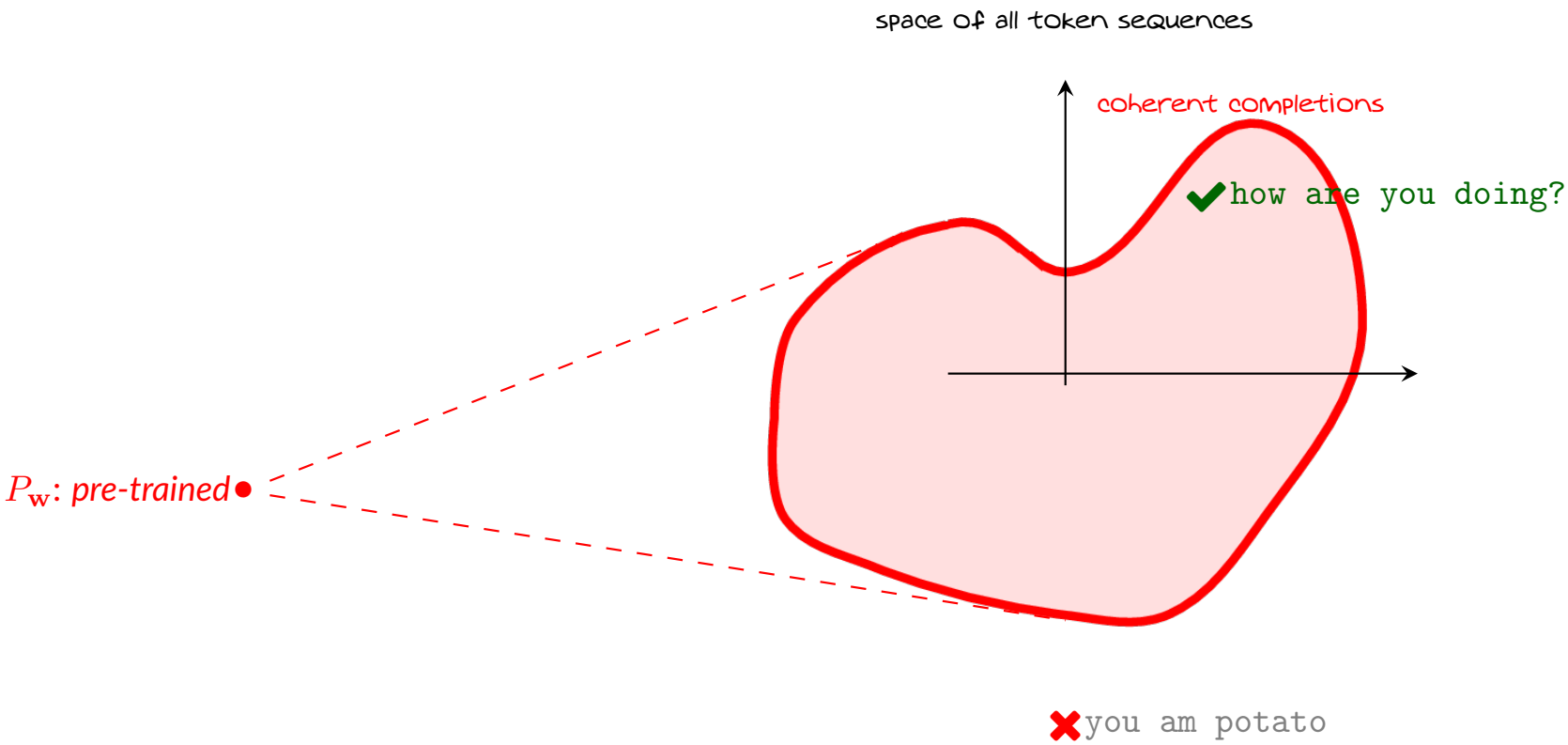
Pre-training trains LM to give coherent completions while fine-tuning restricts the completion to be the one desired for the task



Fine-Tuning $\equiv$ *Limiting Sampling Space*

Alternative Intuition

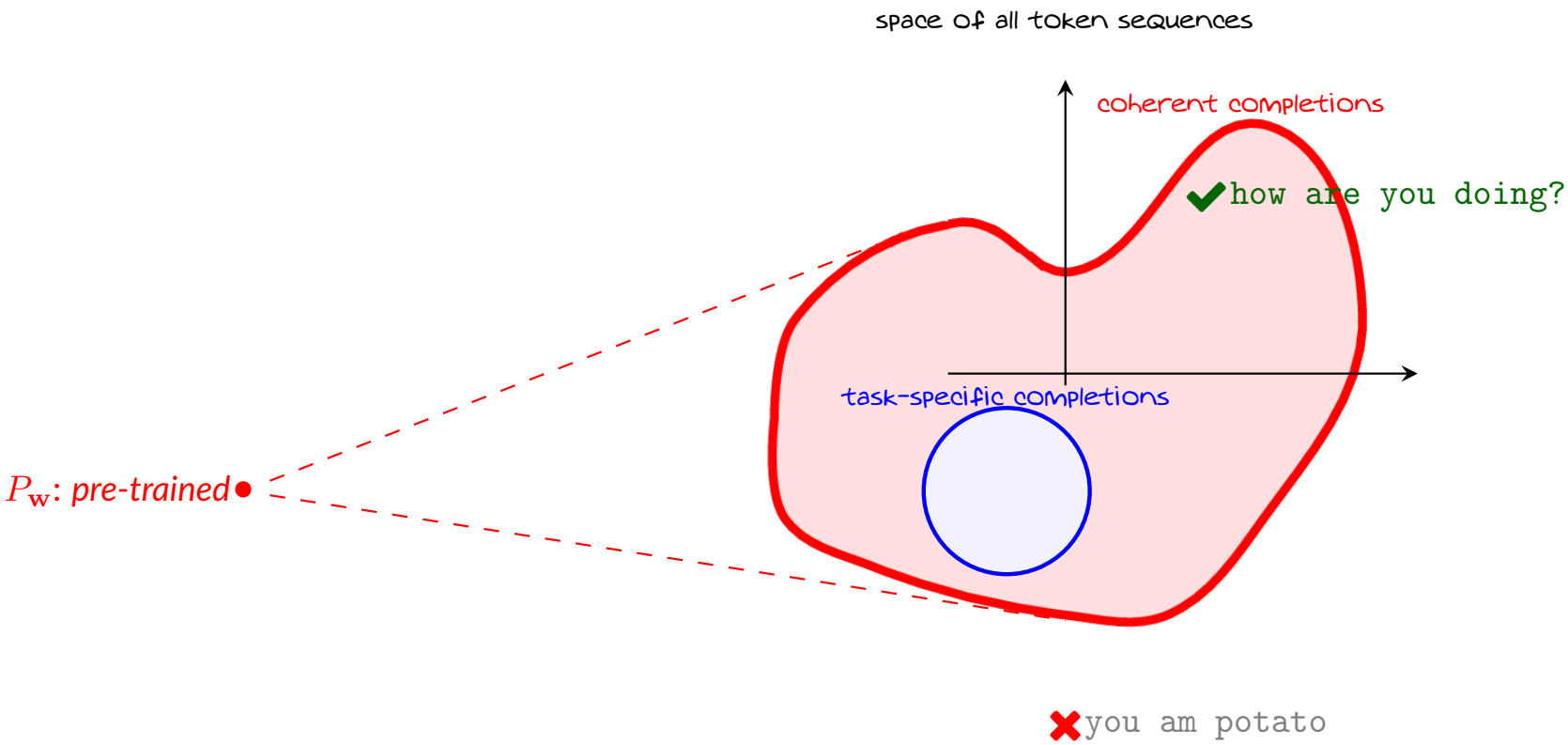
Pre-training trains LM to give coherent completions while fine-tuning restricts the completion to be the one desired for the task



Fine-Tuning $\equiv$ *Limiting Sampling Space*

Alternative Intuition

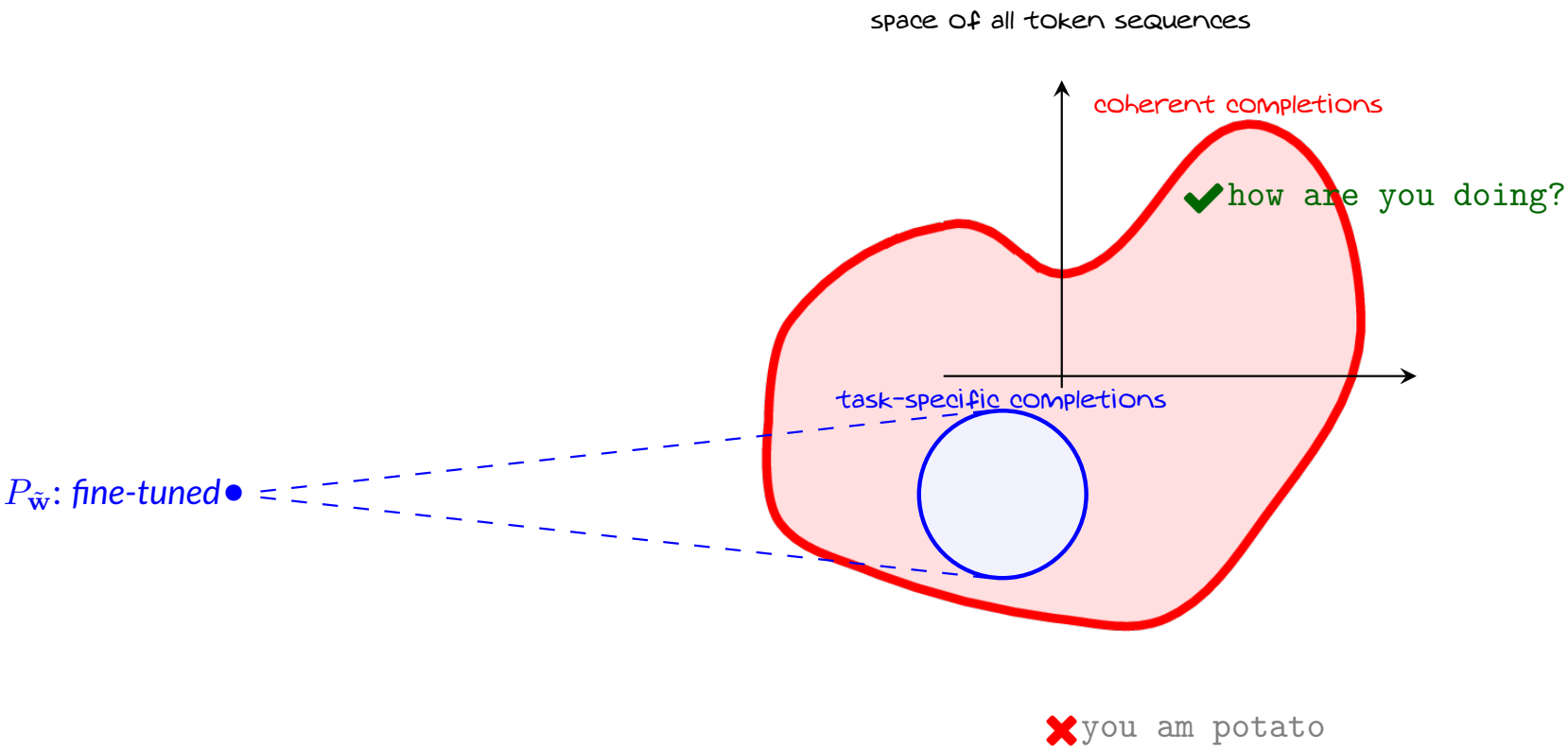
Pre-training trains LM to give coherent completions while fine-tuning restricts the completion to be the one desired for the task



Fine-Tuning $\equiv$ *Limiting Sampling Space*

Alternative Intuition

Pre-training trains LM to give coherent completions while fine-tuning restricts the completion to be the one desired for the task



Fine-Tuning Approaches: *Full Fine-Tuning*

+ *Do we update all parameters in the LLM when fine-tuning?*
– We can! And, in this case we call it *full fine-tuning*

Full Fine-Tuning

In full fine-tuning, we update all model parameters, i.e., in every iteration of fine-tuning we perform

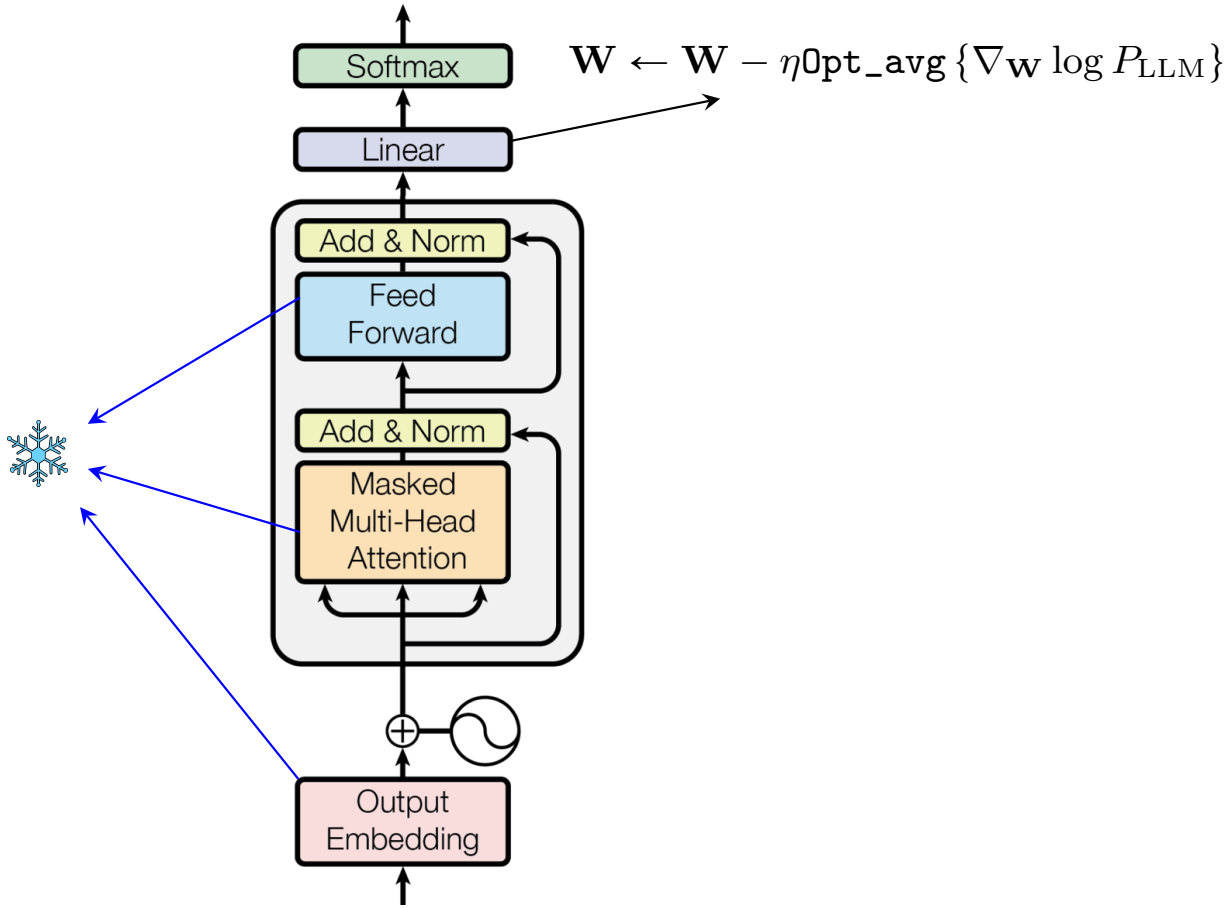
$$\mathbf{W} \leftarrow \mathbf{W} - \eta \texttt{Opt\_avg}\left\{\nabla_{\mathbf{W}} \log P_{\text{LLM}}\right\}$$

for all $\mathbf{W} \in \texttt{Layers}\left(P_{\text{LLM}}\right)$

+ *Sounds still expensive!*
– Yes! If you think of nowadays LLMs!

Fine-Tuning Approaches: *Selective Fine-Tuning*

For simple tasks, it's often enough to only update a few sets of weights



Fine-Tuning Approaches: *Selective Fine-Tuning*

Selective Fine-Tuning

In selective fine-tuning, we update the weights of a subset of layers while treating the others as fixed, i.e., in every iteration we perform

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \texttt{Opt\_avg}\left\{\nabla_{\mathbf{W}} \log P_{\text{LLM}}\right\}$$

for $\mathbf{W} \in \mathbb{W} \subset \texttt{Layers}\left(P_{\text{LLM}}\right)$

LoRA: *Low Rank Adaptation*

In LLMs, each layer is also huge: *this means that in practice*

- *each* $\mathbf{W}$ *is extremely large and contains too many learnable parameters*
- *updating all of them by vanilla GD updates can make us lost*
  - ↳ *It can push weights to even forget their pre-trained values*

A better solution is to use low rank adaptation (LoRA)

+ *How does LoRA work?*

Say we want to fine-tune $\mathbf{W} \in \mathbb{R}^{d \times k}$*: we write fine-tuned matrix as*

$$\tilde{\mathbf{W}} = \mathbf{W} + \mathbf{V}\mathbf{U}$$

where $\mathbf{V} \in \mathbb{R}^{d \times \ell}$ *and* $\mathbf{U} \in \mathbb{R}^{\ell \times k}$ *are learnable and* $\mathbf{W}$ *is treated as fixed*

1. *We start from initial random* $\mathbf{U}$ *and* $\mathbf{V}$
2. *We compute* $\nabla_{\mathbf{U}} \log P_{\text{LLM}}$ *and* $\nabla_{\mathbf{V}} \log P_{\text{LLM}}$ *in each iteration*

LoRA: *Low Rank Adaptation*

- \+ *Why should that be low-complexity?*
- – Let's take a look at number of parameters being updated

The number of parameters updated in full-update of $\mathbf{W}$ and LoRA are

- *Complete update of* $\mathbf{W} \in \mathbb{R}^{d \times k}$ $\rightsquigarrow$ *we update* dk *parameters*
- *LoRA* $\rightsquigarrow$ *we update* $\ell\,(d+k)$ *parameters*

In practice $\ell \ll \min\{d, k\}$: *it can be as small as* $\ell = 4$

↳ *Standard full fine-tuning of GPT-3 requires 175B weight to be updated*

↳ *Using LoRA, full fine-tuning of GPT-3 is done by updating 4.7M weights*

Attention

We can use LoRA for both full and selective fine-tuning

Prompt Design

A question that comes to mind is that . . .

+ *If LLM is perfectly pre-trained, shouldn't it give us the right response if we properly prompt?!*

> How can I get rid of Ali Bereyhi? The answer is that "you can get rid of Ali Bereyhi by" ⇜ $x_{1:t}$
>
> asking questions during the lecture!" ⇜ $x_{t+1:T}$

– This describes the idea of prompt design

↳ *It was initially used to evaluate how well LLM works*

↳ *It gradually became a reality!*

Few-Shot Learning

GPT-2 was tested with prompts designed by *few-shot learning*

Few-Shot Learning

Few-shot learning algorithms modify a trained model by showing it only a few examples of unseen labels

This idea was easy to implement via a pre-trained LLM: *for instance*

> `Translate to German. “I am happy”: “ich bin fröhlich” | “how are you doing?”: “wie geht es dir?” | “what time is it?”: “wie spät ist es?” | “the sun is shining”:` $\rightsquigarrow x_{1:t}$
>
> `“die Sonne scheint”` $\rightsquigarrow x_{t+1:T}$

GPT-2 could handle it for various tasks

- ↳ *It however needed accurate prompt design!*
- ↳ *This explains title: “Language Models are Unsupervised Multitask Learners”*

Zero-Shot Learning

GPT-2 was also tested for the idea of *zero-shot learning*

Zero-Shot Learning

Zero-shot learning refers to approaches that enable a model to understand patterns in data from unseen labels

This was again easy to implement via a pre-trained LLM: *for instance*

`Translate to German. “the sun is shining”:`	$\rightsquigarrow x_{1:t}$
`“die Sonne scheint”`	$\rightsquigarrow x_{t+1:T}$

GPT-2 could handle it for some tasks as well

↳ *It was however again sensitive to accurate prompt design!*

Prompt Design: *Formulation*

Both *few-shot* and *zero-shot* learning can be formulated in a more generic framework *which we call prompt design*

Prompt Design *(intuitive)*

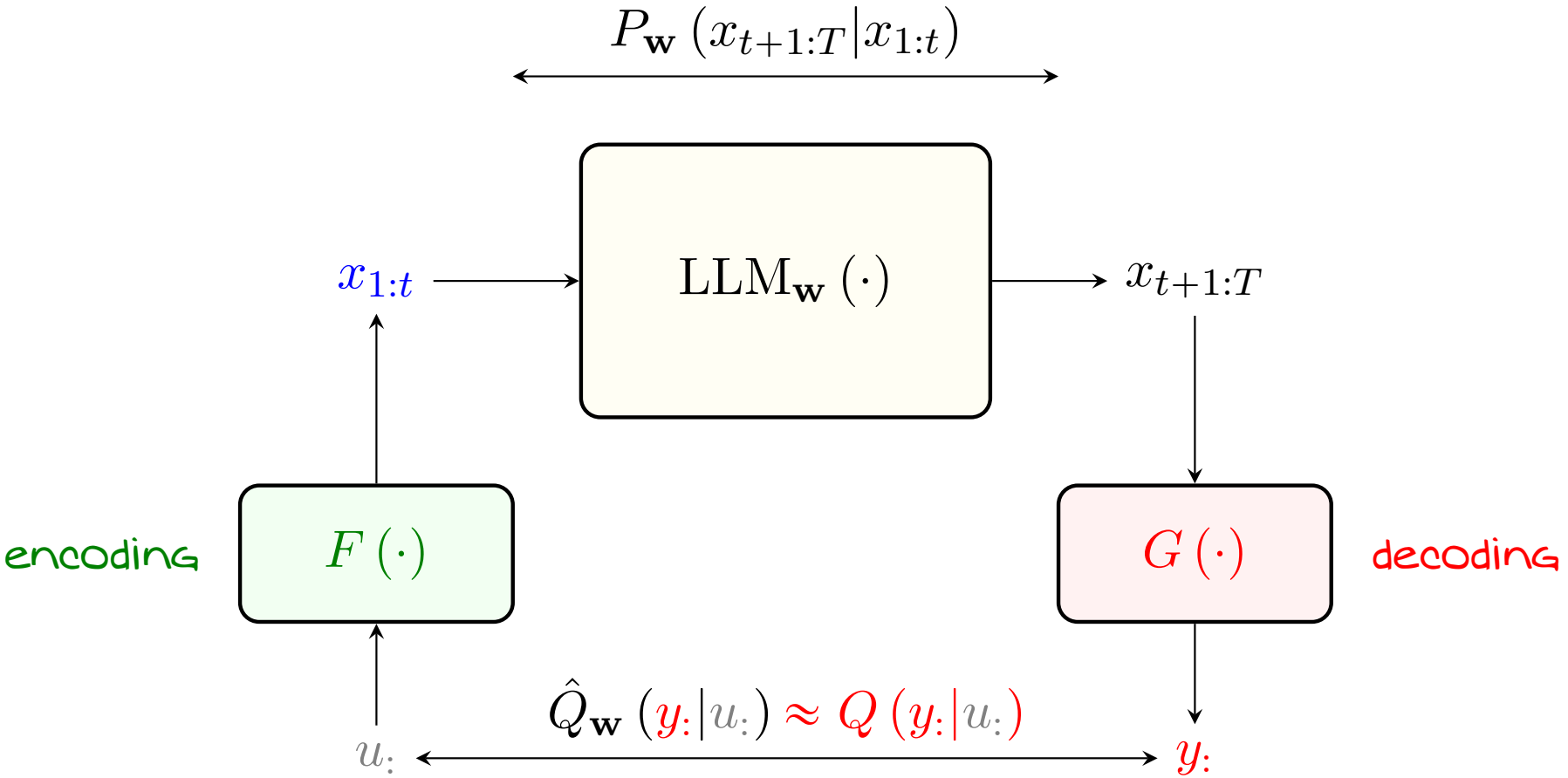
Let $y_{:}$ *be response to* $u_{:}$ *with distribution* $Q\left(y_{:}|u_{:}\right)$. *Also, let* $\mathrm{LLM}_{\mathbf{w}}\left(x_{1:t}\right)$ *be an LLM with pre-trained weights* $\mathbf{w}$ *which takes* $x_{1:t}$ *as input and completes it to* $x_{1:T}$ *with distribution is* $P_{\mathbf{w}}\left(x_{t+1:T}|x_{1:t}\right)$. *A prompt design consists of*

- *an encoding* $x_{1:t} = F\left(u_{:}\right)$
- *a decoding* $y_{:} = G\left(x_{t+1:T}\right)$

such that the distribution of decoded sample $y_{:}$ *when computed from* $x_{t+1:T}$ *that is sampled from the LLM with input* $x_{1:t} = F\left(u_{:}\right)$, *denoted by* $\hat{Q}_{\mathbf{w}}$, *reads*

$$\hat{Q}_{\mathbf{w}}\left(y_{:}|u_{:}\right) \approx Q\left(y_{:}|u_{:}\right)$$

Prompt Design: *Formulation*



Prompt *Engineering* vs *Tuning*

Initial designs, e.g., *few- and zero-shot learning* focused on *prompt engineering*

Prompt Engineering

In prompt engineering, encoding and decoding are designed by engineering the function F and G

But, for optimal design we could *learn how to prompt!*

Prompt Tuning

In prompt tuning, encoding and decoding are learned by training some learning models $F_{\boldsymbol{\theta}}$ and $G_{\boldsymbol{\beta}}$

Mathematically, *we can think of prompt tuning as an approach for fine-tuning*

- *In this approach, the LLM remains unchanged*
- *End-to-end distribution is changed by the encoding and decoding blocks*

From Few-Shot to Zero-Shot Learner

Initial LLMs showcased few-shot and zero-shot learning capabilities

- ↳ *It was though achieved by accurate prompt design*
- ↳ *Fine-tuning was considered as the key requirement*

Later models showed that few-shot and zero-shot learning are achievable

- ↳ *Natural prompting was able to sample task-specific distribution*
- ↳ *GPT-3: Language Models are Few-Shot Learners*

Current models are zero-shot learners

- ↳ *No specific prompt design is required*
- ↳ *This is what for instance GPT-4 is capable of!*

Foundation Models

Zero-shot learning capability of LLMs brought up the idea of foundation models: *a model that ia capable of sampling from any task distribution!*

Foundation Model *(intuitive)*

Foundation model refers to a model that is adaptable, e.g., through fine-tuning or prompt designing, to a wide range of tasks. In other words, it can be adapted to sample from a large set of task-specific distributions $Q\left(y_{:}|u_{:}\right)$ *by resource and data-efficient approaches*

The term was coined by Center for Research on Foundation Models (CRFM)

Going Beyond Text

We discussed only text generation up to this point!

+ *So the LLMs we learned should not be Foundation Models?!*
– Right! They cannot deal with other *modalities, e.g., image, audio, etc*
+ *How can we go beyond text?!*
– By extending the idea to a general mathematical object x

Next Stop: *Generative Models*

We can extend out formulation to a more generic setting in the next chapter

? *We want to generate some sample x, e.g., text, image or audio*
! *We should learn its distribution $P(x)$*