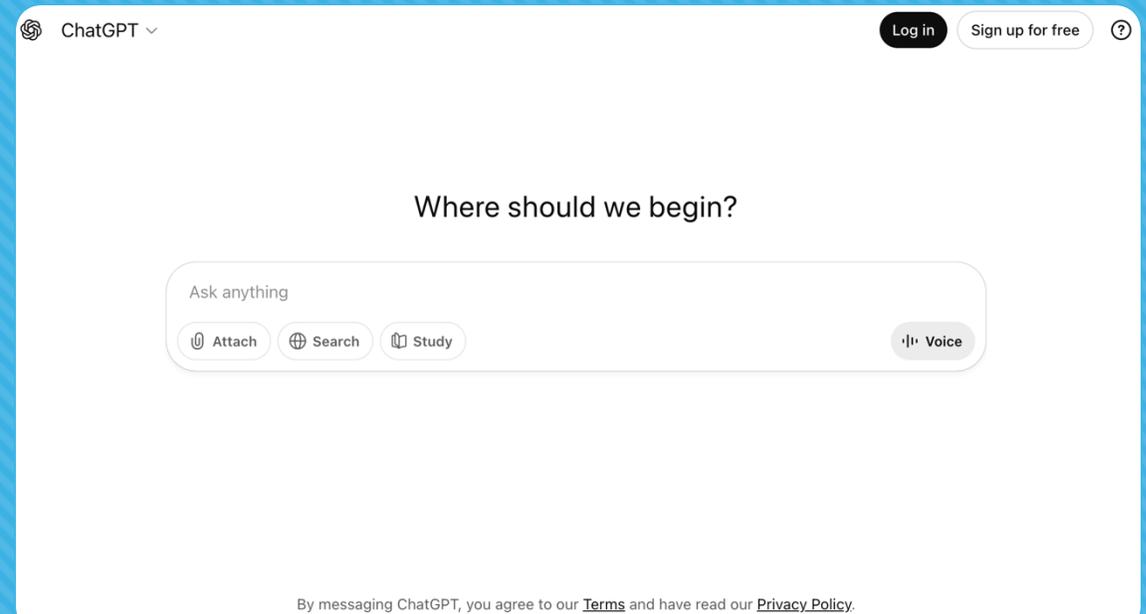
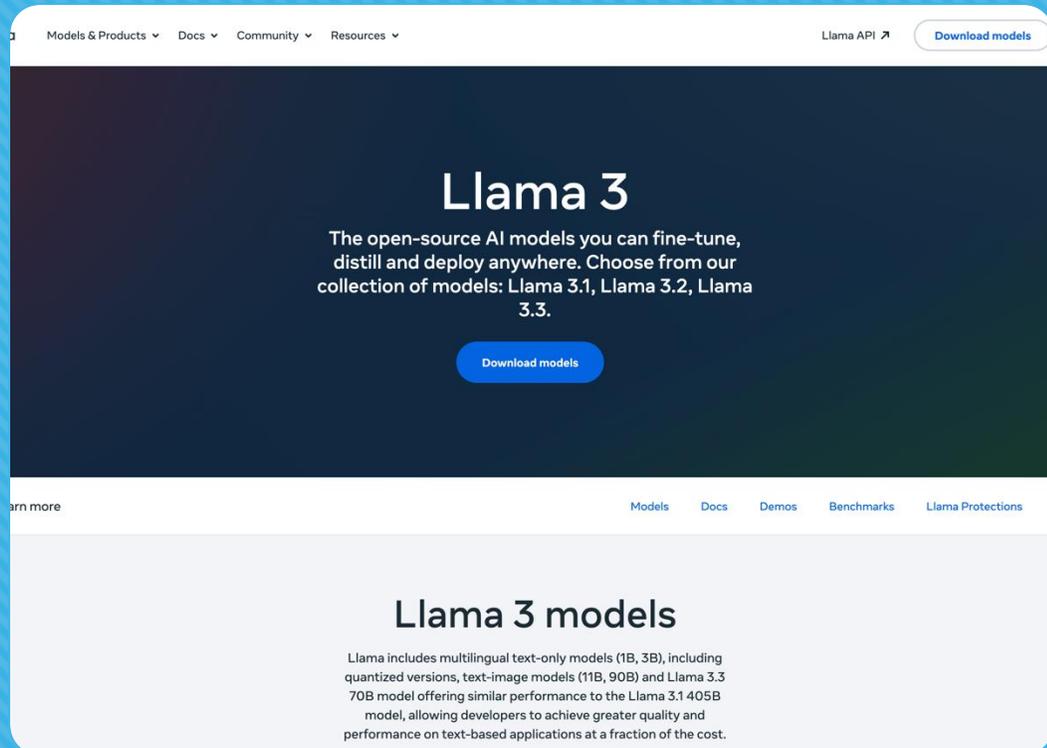


Generative AI

Natalie Parde

UIC CS 421



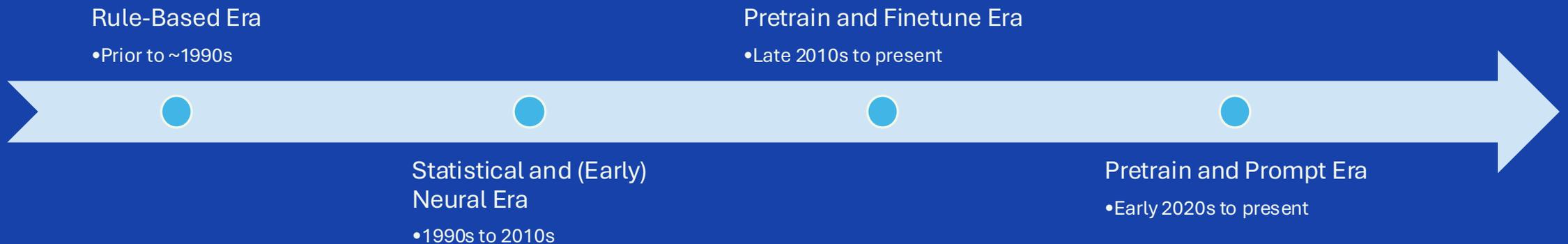
Many modern NLP approaches are implemented using a type of deep learning often referred to as generative AI.

What is generative AI?

- AI models that generate data!
- Generative AI for NLP problems relies on **large language models (LLMs)**
- Recall the distributional hypothesis: we can learn a word's meaning by understanding the contexts in which it typically occurs
- This can be done using n-gram language modeling, but it can often be done more effectively using neural network approaches
- A large language model is a neural language model that has been pretrained on a large amount of text

Recent advancements in generative AI have ushered in new training paradigms.

- Researchers began to consider task formulations in which they could finetune a pretrained model for a new purpose, rather than training a smaller model for that purpose from scratch
- They also began to examine task formulations where they could use pretrained models directly



This Week's Topics



Large Language Models
Prompting
Fine-Tuning

Thursday

Tuesday

Masked Language Modeling
Popular Bidirectional
Transformer Architectures
More Fine-Tuning
Best Practices for Data-
Driven Learning

How can we build a large language model?

- LLMs generally use the Transformer architecture
 - Most LLMs will have multiple Transformer blocks with multi-head attention
 - Following the Transformer block(s), LLMs will have a language modeling head
 - Task/classifier head designed to predict which word comes next

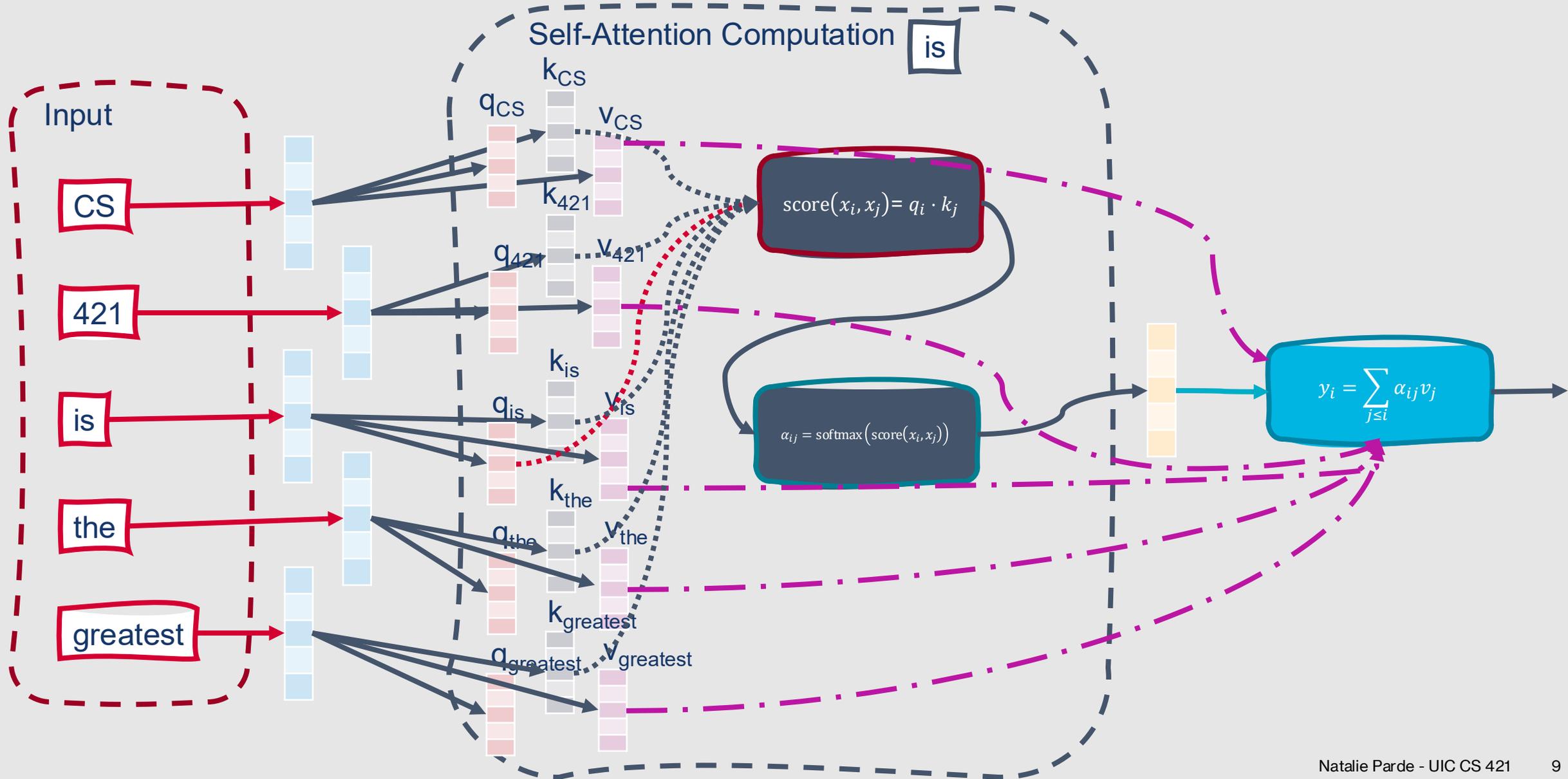
Recall: Transformers

- General premise:
 - Deep learning models don't need to wait to process items one after the other to incorporate sequential information
- Classic feedforward neural network:
 - Input to a layer is a vector of numbers representing the outputs of all units in the previous layer
- Modification for recurrent neural networks:
 - Input to a layer is a vector of numbers representing the outputs of all units in the previous layer + a vector of numbers representing the layer's output at the previous timestep
- Modification for Transformers:
 - Input to a feedforward layer is the output from a **self-attention layer** computed over the entire input sequence, indicating which words in the sequence are most important to one another

Self-Attention

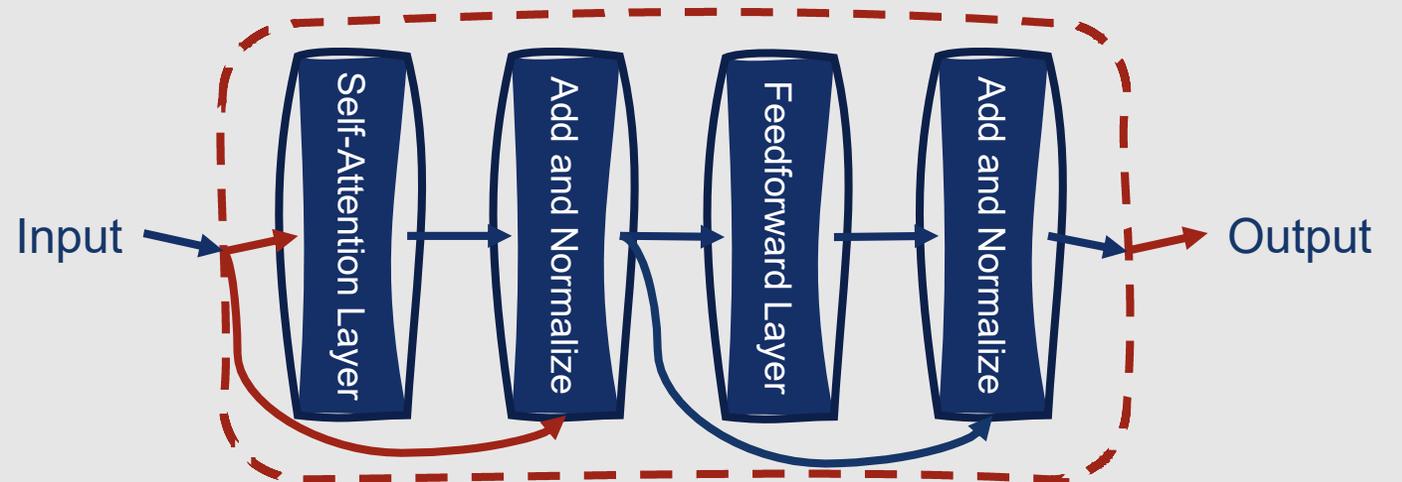
1. Generate key, query, and value embeddings for each element of the input vector \mathbf{x}
 - $\mathbf{q}_i = \mathbf{W}^Q \mathbf{x}_i$
 - $\mathbf{k}_i = \mathbf{W}^K \mathbf{x}_i$
 - $\mathbf{v}_i = \mathbf{W}^V \mathbf{x}_i$
2. Compute attention weights α by applying a softmax activation over the element-wise comparison scores between all possible query-key pairs in the full input sequence
 - $\text{score}_{ij} = \mathbf{q}_i \cdot \mathbf{k}_j$
 - $\alpha_{ij} = \frac{\exp(\text{score}_{ij})}{\sum_{k=1}^n \exp(\text{score}_{ik})}$
3. Compute the output vector \mathbf{y}_i as the attention-weighted sum of the input value vectors \mathbf{v}
 - $\mathbf{y}_i = \sum_{j=1}^n \alpha_{ij} \mathbf{v}_j$

Bidirectional Self-Attention Layer

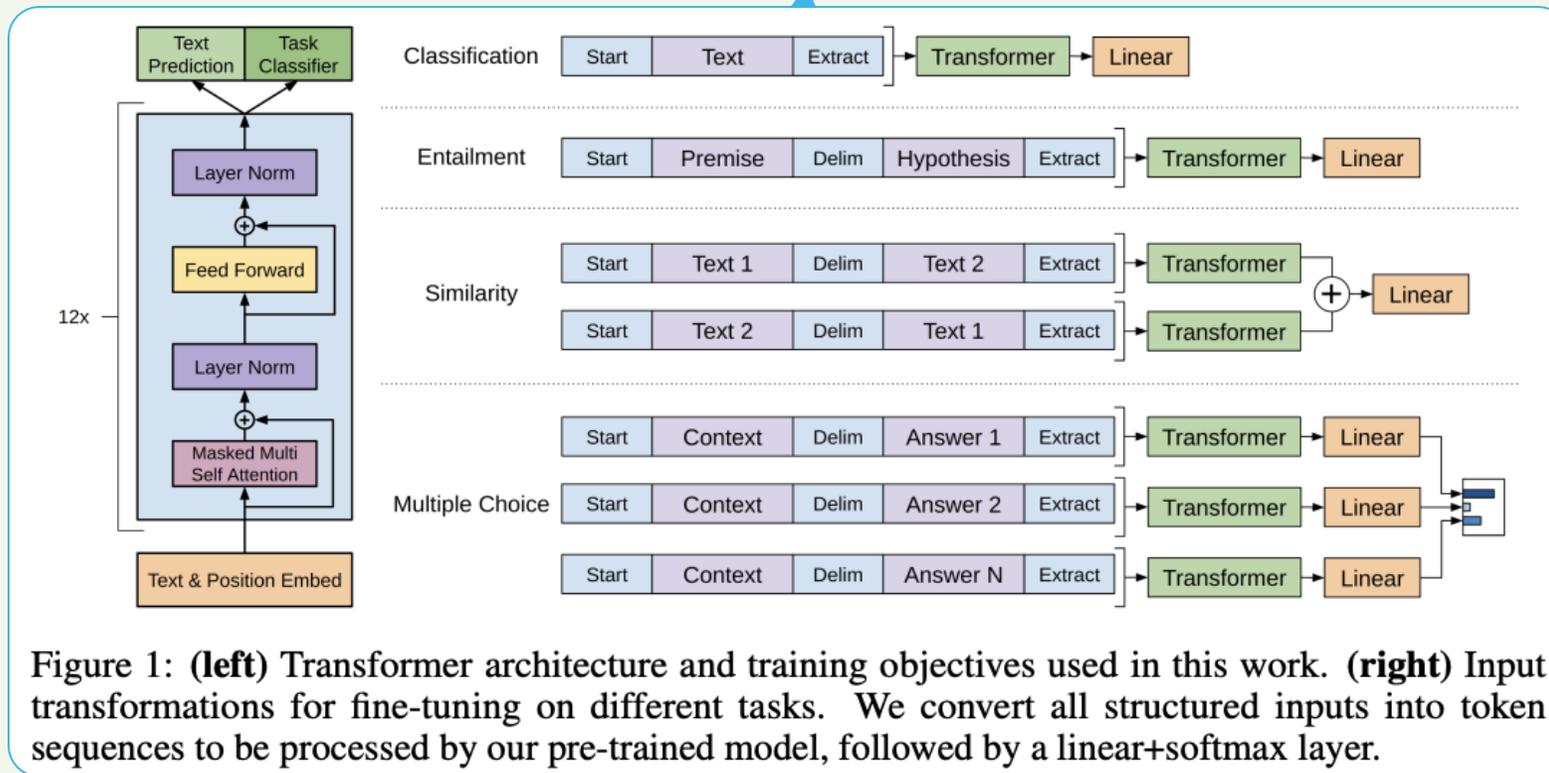


Transformer Blocks

- Transformers are implemented by stacking one or more blocks of the following layers:
 - Self-attention layer
 - Normalization layer
 - Feedforward layer
 - Another normalization layer
- Some of these layers have **residual** connections between them even though they do not immediately precede or proceed one another



We then implement large language models by stacking Transformer blocks!



Radford, Alec, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. "Improving Language Understanding by Generative Pre-Training." OpenAI Report. 2018. https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

LLM Pretraining

- “GPT-Style” LLMs are pretrained to perform next word prediction
- Self-supervised training algorithm
 - Take a very large corpus of text as training material (no manual labels required)
 - At each time step, learn to predict the next word in the training corpus
 - Minimize the error in predicting the true next word using cross-entropy loss

Cross-Entropy Loss for LLM Training

- As we've seen previously, cross-entropy measures the difference between predicted and true probability distributions
 - $L_{CE} = -\sum_{w \in V} \mathbf{y}_t[w] \log \hat{\mathbf{y}}_t[w]$
- In language modeling, our correct distribution \mathbf{y}_t comes from knowing what the actual next word is
- We create a **one-hot encoding** where the dimension for the actual next word has a value of 1 and all other dimensions have values of 0
- This means that in language modeling, **cross-entropy loss will be governed by the probability the model assigns to the correct next word** (everything else gets multiplied by values of 0!), so we can simplify cross-entropy loss at a timestep t to:
 - $L_{CE}(\hat{\mathbf{y}}_t, \mathbf{y}_t) = -\log \hat{\mathbf{y}}_t[w_{t+1}]$

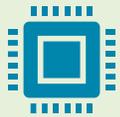
Teacher Forcing

- Most LLMs are also pretrained using **teacher forcing**
- **Teacher forcing:** Always give the model the correct (rather than predicted) history sequence to predict the next word
 - At each word position of the input, take the correct sequence of tokens and use them to compute a probability distribution over the possible next words, thereby computing the model's loss for the next token
 - Move to the next word, ignore (for training purposes) what the model just predicted, and instead use the correct sequence of tokens as input once again to estimate the next word



Overall, this means that pretraining LLMs is done by...

- At a particular timestep, given all the preceding words, use the final layer to produce an **output distribution over the full vocabulary**
- During training, use the probability assigned to the correct word to calculate **cross-entropy loss**
 - To get the loss for an entire training sequence, average cross-entropy loss across the sequence
- Adjust the weights in the network to minimize cross-entropy loss over the entire training sequence via **gradient descent**
- Note that Transformers allow for the entire sequence to be processed in parallel since the output for each element is computed separately (in contrast to RNNs, which requires serial calculation due to the recurrence in hidden states)



Since Transformers process an entire input sequence in parallel, they require a fixed context window

Much larger than context windows seen in n-gram language models!

GPT-4 has a context window of 4096 tokens

Llama 3 has a context window of 8192



What if our text is shorter or longer than the context window?

Shorter: Pad the input with zeroes or input multiple texts in a single context window, separated by end-of-text tokens

Longer: Truncate the text (but think carefully about how to do so and whether this will result in loss of useful information!)

How to handle “fixed” sequence lengths in Transformers?

Training Corpora for LLMs

- Mainly trained on large corpora scraped from the web:
 - Common Crawl: <https://commoncrawl.org/>
 - Colossal Clean Crawled Corpus (C4): <https://github.com/allenai/allennlp/discussions/5056>
 - Subset of Common Crawl (156 billion English-language tokens) that has been deduplicated, filtered for non-natural language (e.g., code), and filtered for offensive words
 - Dolma: <https://allenai.github.io/dolma/>

When using raw data from the web, it's a best practice to filter for quality and safety.

- **Quality Filters:** Train classifiers to assign quality scores to documents
 - Very subjective, so often multiple quality filters should be considered
 - For example, you may want quality filters to:
 - Prioritize high-value reference corpora (e.g., sources that you deem reliable)
 - Avoid websites with personal identifiable information (PII)
 - Avoid websites with offensive or restricted content
 - You can also implement quality filters to:
 - Remove boilerplate text (very common online!)
 - Deduplicate, or remove duplicate documents
- In addition to helping control what data is used to train your model, quality filtering often improves language model performance

Safety Filtering

- Very subjective (desired safety filters may vary for people from different geographic, social, or cultural groups)
- Often includes **toxicity detection**, which classifies text based on whether it employs pejorative or stigmatizing language
 - Doesn't tend to work as well in data generated by speakers of minoritized dialects
 - Tends to *require* toxic language in order to learn to detect toxicity



Legal and Ethical Issues Regarding LLM Pretraining Data

- **Copyright law** varies across countries, and large web corpora contain a large amount of copyrighted content
 - In the US, active legal discussions focus on whether the fair use doctrine extends to LLMs that are used to generate text that compete with the market from which they are trained!
- **Data consent** for large web corpora is often unclear or even ignored during the scraping process
 - Questions exist regarding the extent to which Terms of Service and robots.txt files are legally valid across countries, and the extent to which new restrictions may be applied retroactively
- **Privacy issues** may arise when filters are not in place or fail to remove private information such as phone numbers or IP addresses

This Week's Topics



Large Language Models
Prompting
Fine-Tuning

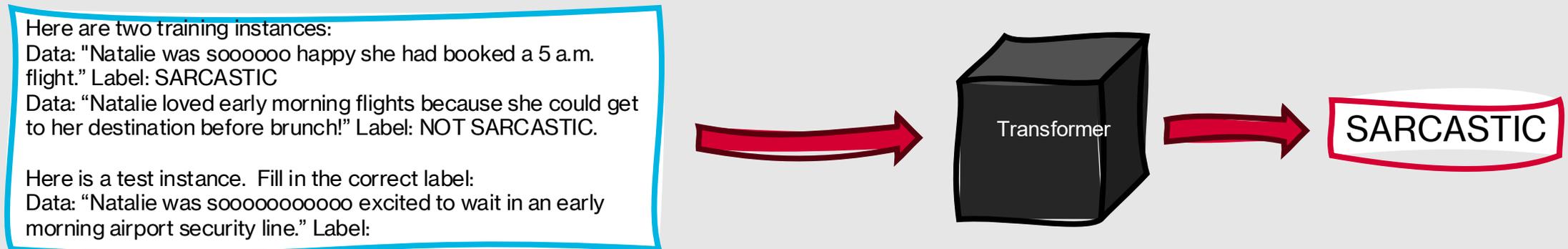
Thursday

Tuesday

Masked Language Modeling
Popular Bidirectional
Transformer Architectures
More Fine-Tuning
Best Practices for Data-
Driven Learning

Pretrain and Prompt Paradigm

- Intuition:
 - If we take extremely large language models that have been pretrained to predict the next word on a wide variety of language data, we can **prompt** them to produce labels or output for new tasks
- Popular pretrained model for this purpose: GPT



Prompting: Using Pretrained LLMs to Solve NLP Tasks

- **Conditional generation:** Generate text conditioned on an input text string
 - The input text string or context is often referred to as a prompt
- Large Transformer-based models work well for conditional generation because they facilitate consideration of very large contexts
 - RNNs can theoretically consider infinite context, but in practice they tend to focus on the closest context and minimize the influence of long-range dependencies
 - Transformers may require a fixed-length context window for practical purposes (e.g., training them on a particular GPU cluster) but the context window is *much* larger than what would be required for n-gram language modeling or feedforward neural language models

How does prompting work?

- Take a large model that has already been trained to perform generative language modeling
- Develop a set of **prompt templates** for your task
 - Prompt templates can be manually or automatically constructed
- Develop an approach for **answer engineering**
 - Build an answer space (set of possible answers that your model may generate) and map that answer space to your desired outputs
 - This can also be done manually or automatically using search techniques
- Format your input according to the relevant prompt template(s) and map the resulting language model output to your desired target output

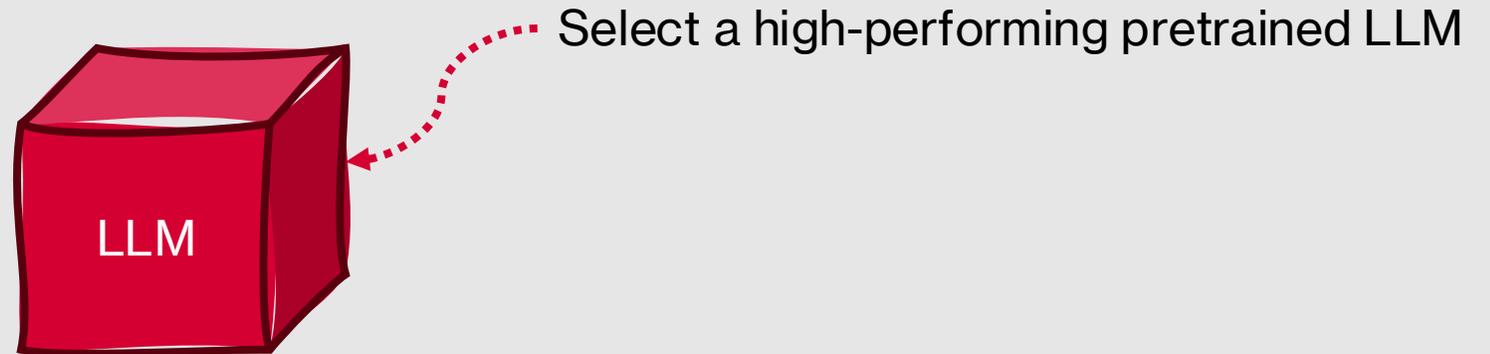
Common Prompting Strategies

- Instruction-Based Prompting
 - Detailed prompts with context help the model to generate focused, relevant content
 - Specifying the desired output format, length, and tone helps the model generate output that meets expectations
- Role-Based Prompting
 - Specify role(s) for the model to fill to help tailor responses with the desired tone and level of expertise
- Few-Shot Prompting
 - Use examples to demonstrate the desired input-output transformation for new inputs
- Chain-of-Thought Prompting
 - Specify that the model should reason step by step, increasing its likelihood of generating logically correct output

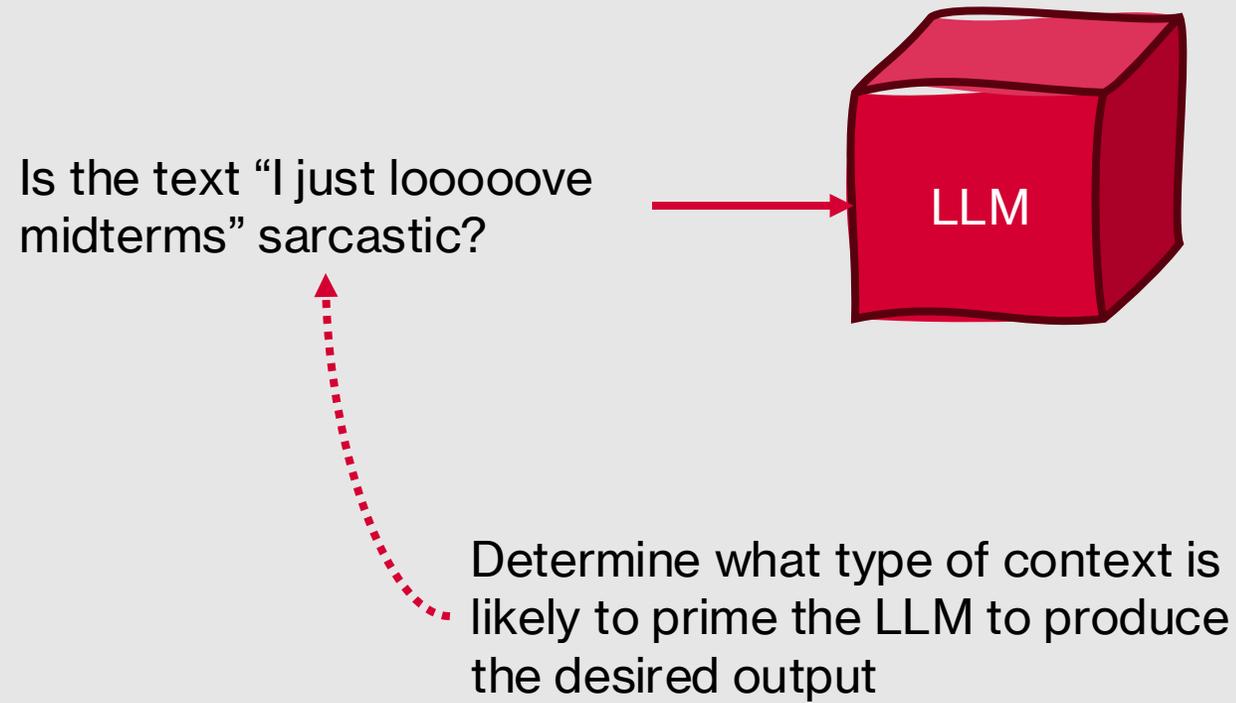
Additional Prompting Strategies

- Structured Output Prompting
 - Specify structured output formats, such as JSON files or tables, to improve output consistency
- Prompt Chaining
 - Break the task into subtasks and use the output from an earlier task as the input for the next task, creating a “chain” of prompts

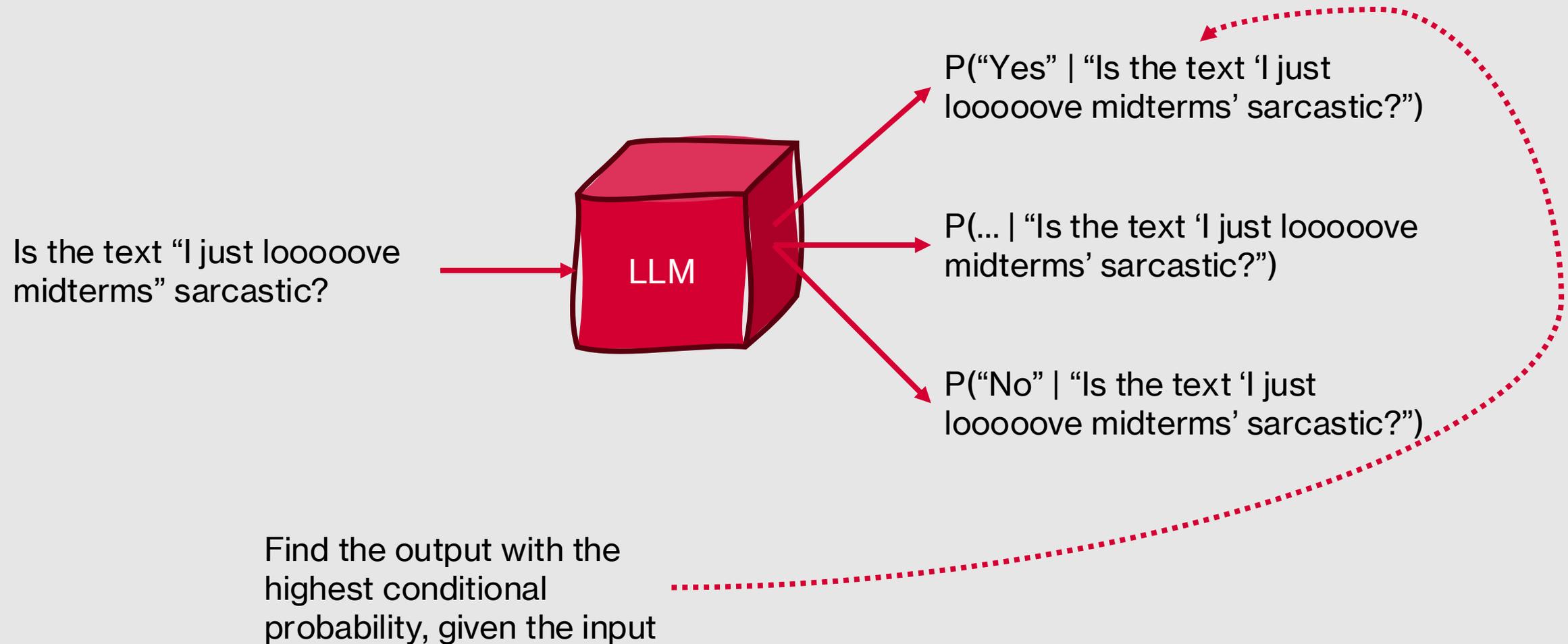
Case Example: Zero-Shot Task “Learning”



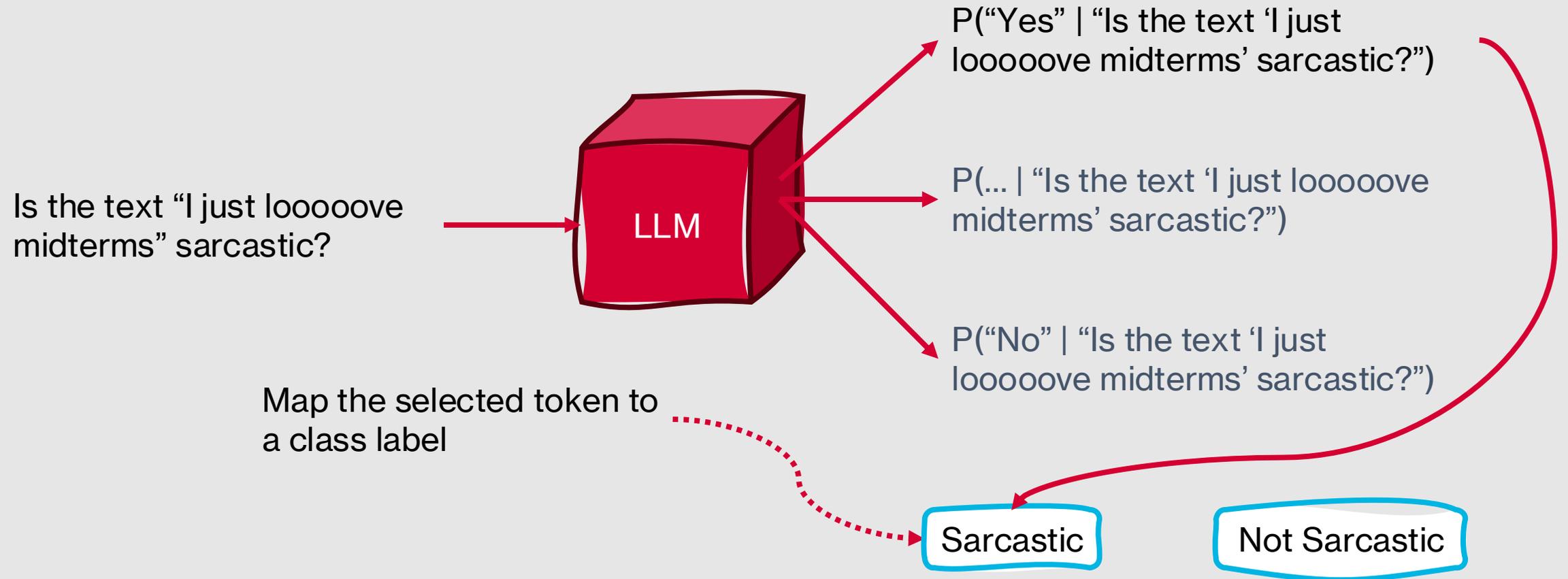
Case Example: Zero-Shot Task “Learning”



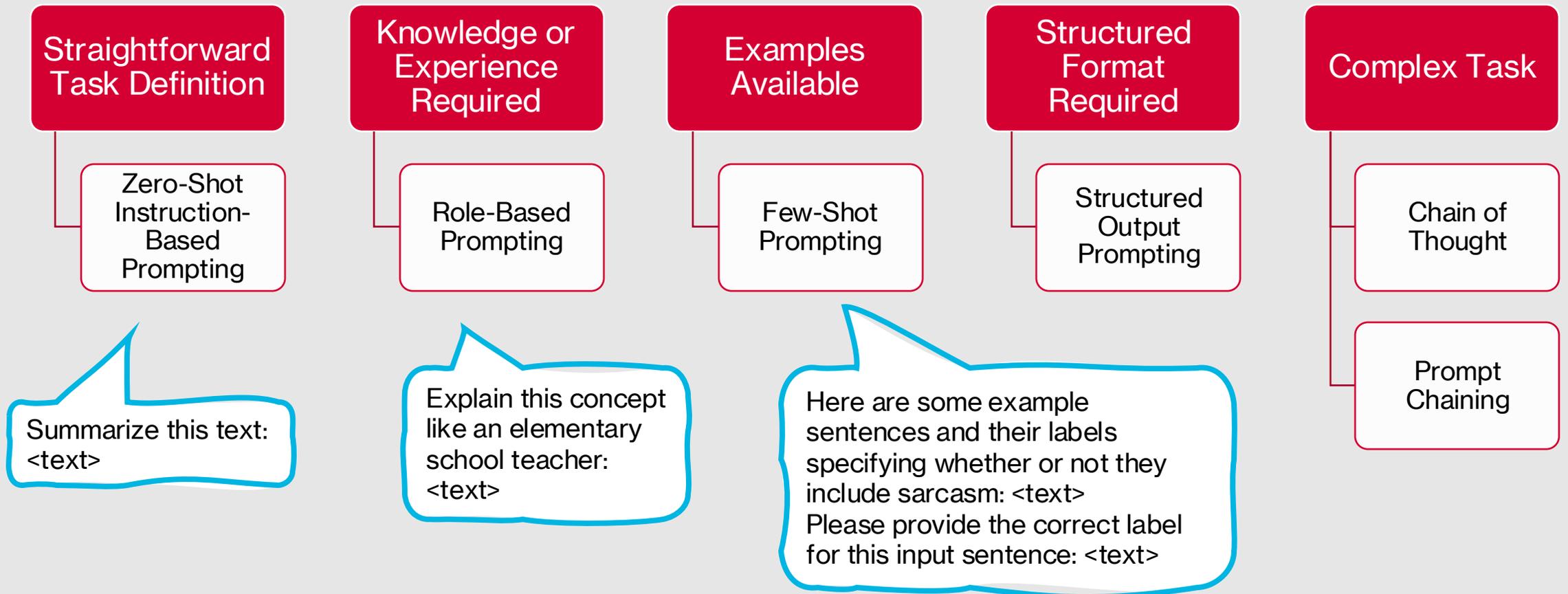
Case Example: Zero-Shot Task “Learning”



Case Example: Zero-Shot Task “Learning”

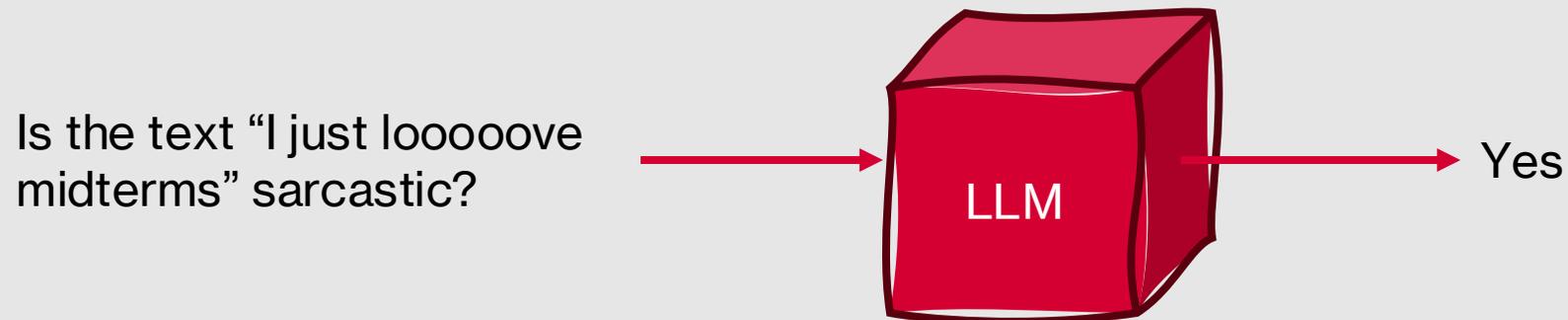


When to use different prompting strategies?



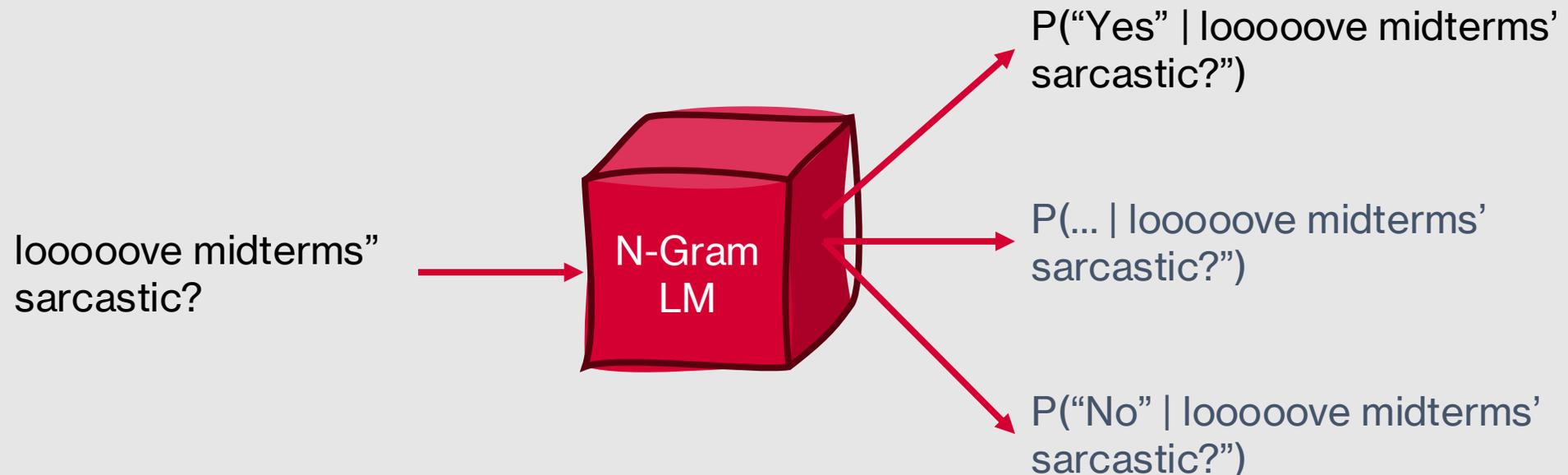
Why solve complex NLP tasks using techniques like this?

- LLMs are very powerful due to the amount of contextual information that they learn during pretraining
- Many NLP tasks can take advantage of this contextual knowledge if they are cast as word prediction tasks
- In some cases, this can be done effectively with no task-specific training at all



Context window is key....

- The long context window used in popular Transformer-based LLMs is key to facilitating classification tasks in this scenario
- N-gram and feedforward neural language models: context isn't large enough to capture full task description and/or input sample



We can solve a wide variety of NLP tasks using prompting!

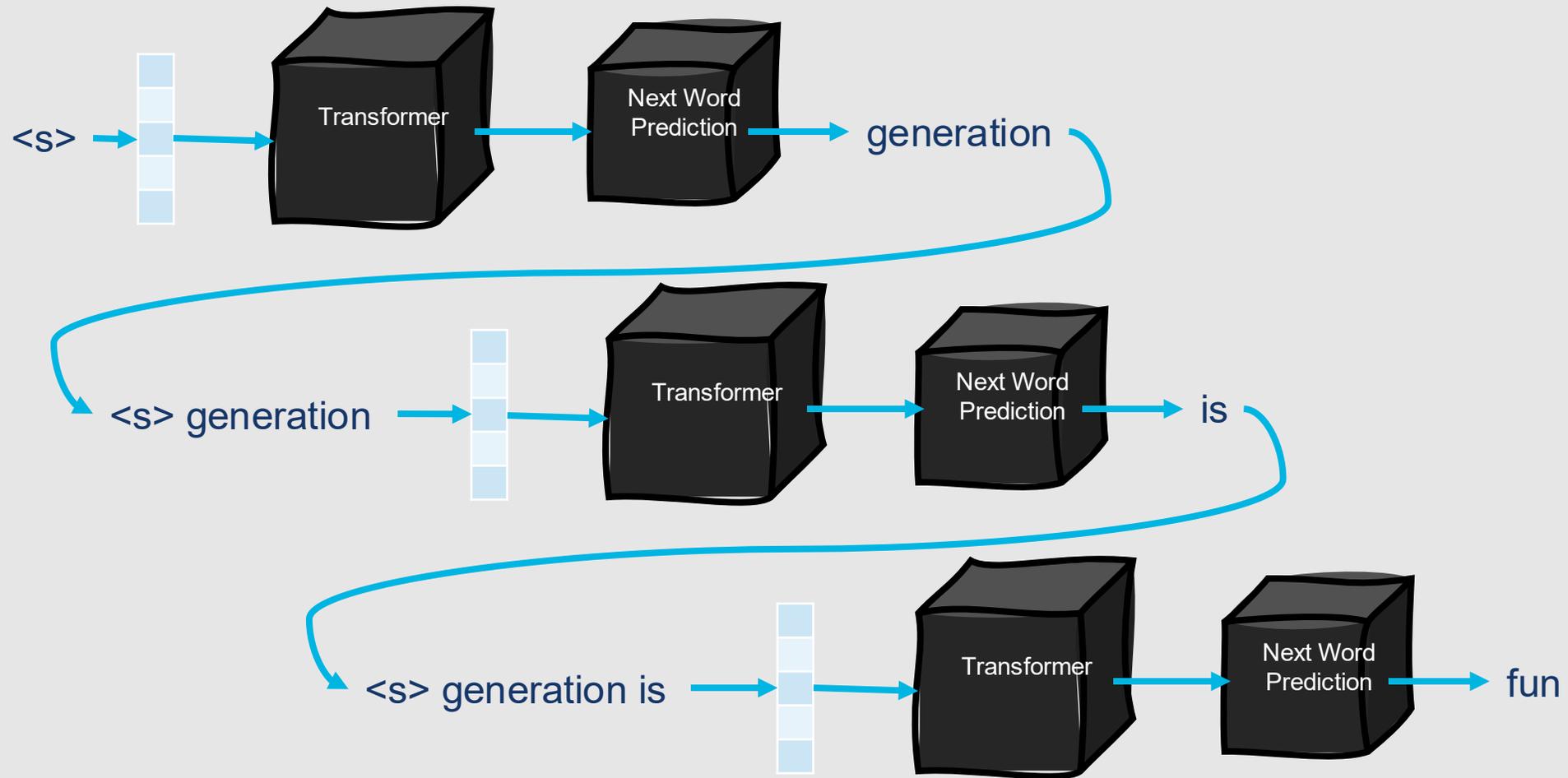
Question answering

- Context is the question
- Context-conditioned output is the answer
 - Rationale: Words that are contextually likely to be generated following a question often form the correct answer

Summarization

- Context is a long passage of text, followed by token(s) that often indicate that a summary is desired in large web-based corpora
 - Popular token for this: “tl;dr”
- Context-conditioned output is a summary
 - Rationale: Certain signals indicating that a summarization follows are popular enough in web-based corpora that the language model learns to perform this as a text completion

In general, the process of generating long-form output using LLMs is called **autoregressive generation**.



Autoregressive Generation

- Also sometimes called **causal language model generation**
- Decodes a language model *only in one direction* (i.e., from the beginning of the text to the end of the text)



Autoregressive Generation Strategies

- **Greedy decoding:** Generate the most likely word, given the context
 - $\hat{w}_t = \operatorname{argmax}_{w \in V} P(w | \mathbf{w}_{<t})$
 - Produces locally optimal solutions
 - When generating longer sequences of text, may not ultimately result in the best output
- Common issues with greedy decoding:
 - Generated text tends to be generic
 - Generated text tends to be repetitive
 - Identical contexts will produce identical output
 - Determinism is good for replicability, but not necessarily good for generating realistic output

How can we improve upon greedy decoding?

- **Beam search:** Generate numerous possible completions while avoiding generating completions that stray beyond a fixed “beam width” and select the highest-scoring completion after generating all possibilities in full
- Introduce **sampling methods** to diversify generated text: Choose words randomly according to their probabilities assigned by the model

Popular Sampling Strategies for LLMs

- **Random Sampling:** At each timestep, sample words according to their probability conditioned on the previous output until that point
 - $i \leftarrow 1$
 - $w_i \sim p(w)$ # choose w_i by sampling from the probability distribution $p(w)$
 - while $w_i \neq \text{EOS}$:
 - $i \leftarrow i + 1$
 - $w_i \sim p(w_i | w_{<i})$
 - Advantages: Straightforward and easy to implement
 - Disadvantages: Selects rare words somewhat often, resulting in output that can seem “weird”

Important considerations during sampling....

- **Quality:** The generated output is accurate, coherent, and factual
- **Diversity:** The generated output is interesting and unique
- When sampling output from LLMs, quality and diversity are often at odds with one another!
 - Diversity is often improved by selecting slightly less probable words, but this tends to reduce factuality and coherence

Popular Sampling Strategies for LLMs

- **Random Sampling:** At each timestep, sample words according to their probability conditioned on the previous output until that point, and use a Transformer-based language model to calculate that probability
- **Top-k Sampling:** Instead of choosing the most probable word to generate:
 1. Truncate the distribution to the top k most likely words
 2. Renormalize the probability distribution
 3. Randomly sample from among those k words according to the renormalized probabilities
- Tends to produce more diverse text while still retaining reasonably good quality

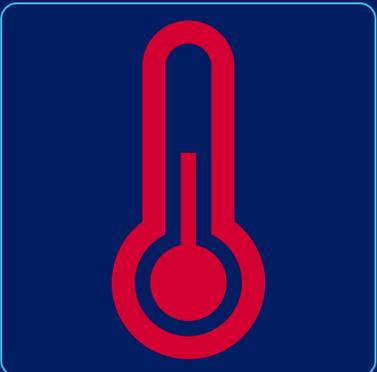
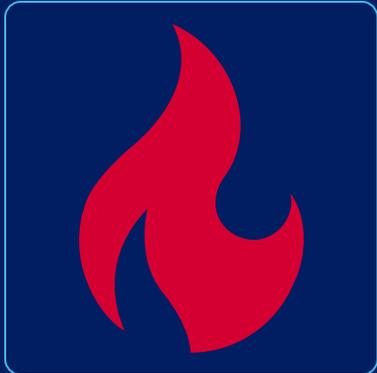
Popular Sampling Strategies for LLMs

- **Random Sampling:** At each timestep, sample words according to their probability conditioned on the previous output until that point, and use a Transformer-based language model to calculate that probability
- **Top-k Sampling:** Instead of choosing the most probable word to generate, sample from among the top k most likely words
- **Nucleus or Top-p Sampling:** Instead of keeping the top fixed-k words, keep the top p percent of the probability mass
 1. Truncate the distribution to remove very unlikely words in terms of their probability (don't keep a predetermined number of words)
 2. Proceed similarly to top-k sampling
 - By dynamically increasing and decreasing the pool of word candidates, this approach may be more robust in different contexts

Popular Sampling Strategies for LLMs

- **Random Sampling:** At each timestep, sample words according to their probability conditioned on the previous output until that point, and use a Transformer-based language model to calculate that probability
- **Top-k Sampling:** Instead of choosing the most probable word to generate, sample from among the top k most likely words
- **Nucleus or Top-p Sampling:** Instead of keeping the top fixed-k words, keep the top p percent of the probability mass
- **Temperature Sampling:** Reshape the distribution rather than truncating it
 - Intuition comes from thermodynamics: Systems at high temperatures are flexible and can explore many possible states, whereas systems at low temperatures focus on the most probable words and decrease the probability of exploring rare words

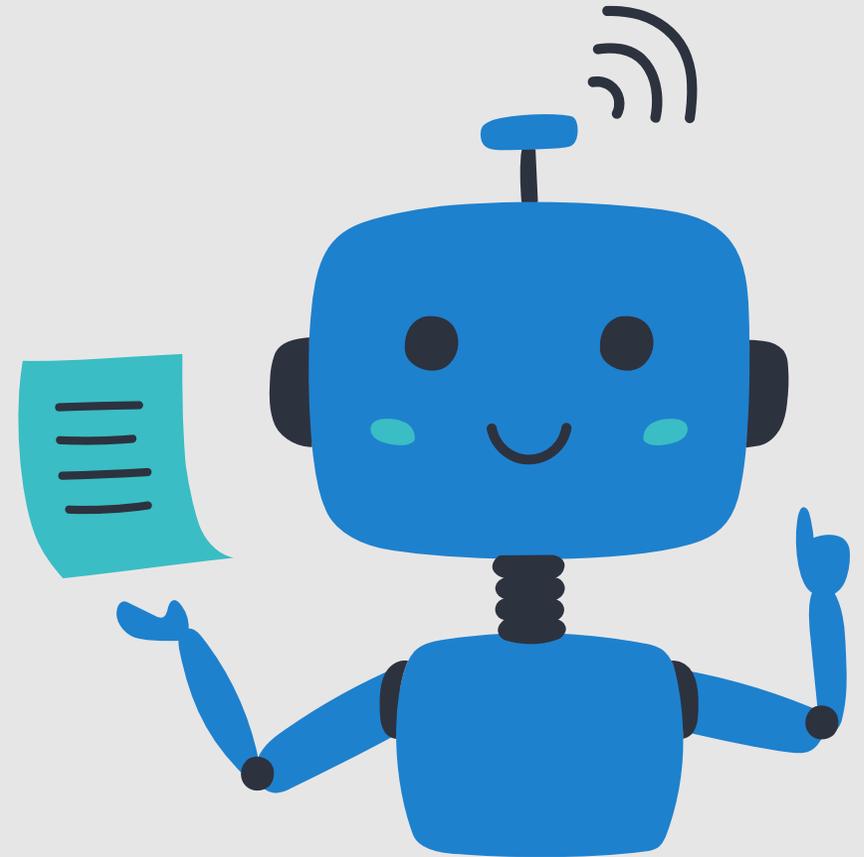
Temperature Sampling



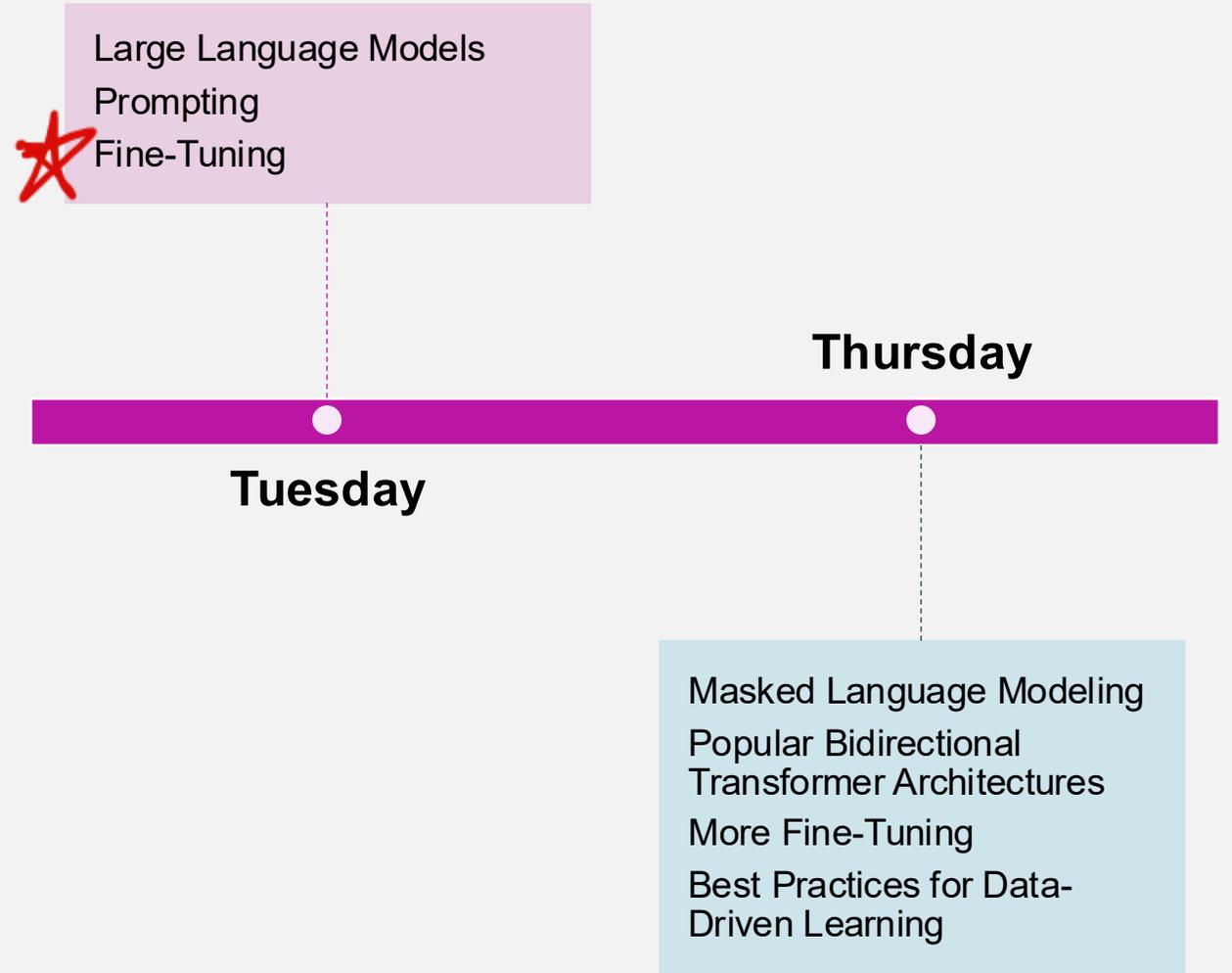
- Divide the logit by a temperature parameter, τ , before normalizing it and passing it through the softmax function
 - $y = \text{softmax}\left(\frac{z}{\tau}\right)$
- Why does temperature sampling work?
 - Consider the low temperature sampling range $\tau \in (0,1]$:
 - When τ is very close to 1, the distribution remains very similar to the original z
 - With lower τ , then $\frac{z}{\tau}$ results in larger values being passed to the softmax function
 - Larger values passed to softmax result in increased probabilities for those corresponding words and accordingly smaller probabilities for the other, lower-probability words
 - As τ approaches 0, the approach grows greedier and the probability of the most likely word approaches 1.0
 - In high temperature sampling ($\tau > 1$), we can progressively flatten the word probability rather than making it greedier

Advantages of Prompting

- Successful **few-shot** or even **zero-shot** approaches facilitate learning (or “learning”) from few or no training examples
- This allows researchers to build models for tasks that were previously inaccessible due to extremely scarce resource availability
- Prompting also requires **limited or no parameter tuning** for the base language model, making it possible to develop classifiers more efficiently



This Week's Topics



What about cases when we *can't* use a pretrained LLM from scratch?

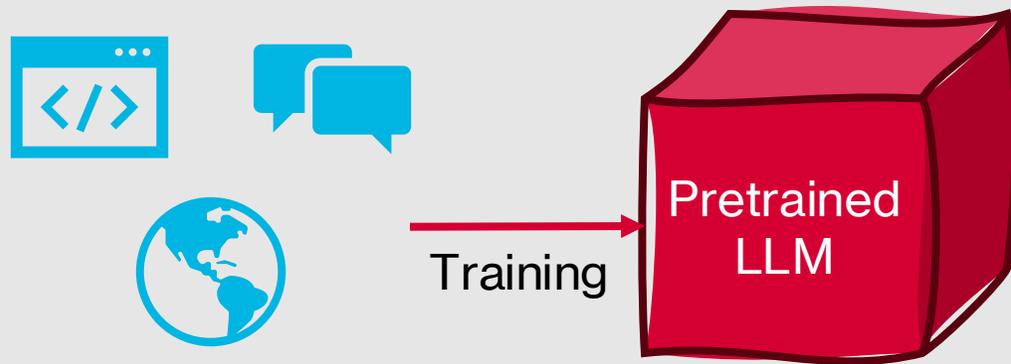
- We can use LLMs as base models and further develop them for specific tasks using a process called fine-tuning
- This may be useful in many specialized settings:
 - Legal text
 - Medical text
 - Low-resource languages

Pretrain and Finetune Paradigm

- Intuition:
 - If we take models that have been **pretrained** on massive datasets for other tasks, we can **finetune** them for our specific task while also taking advantage of the information that was learned during the pretraining process

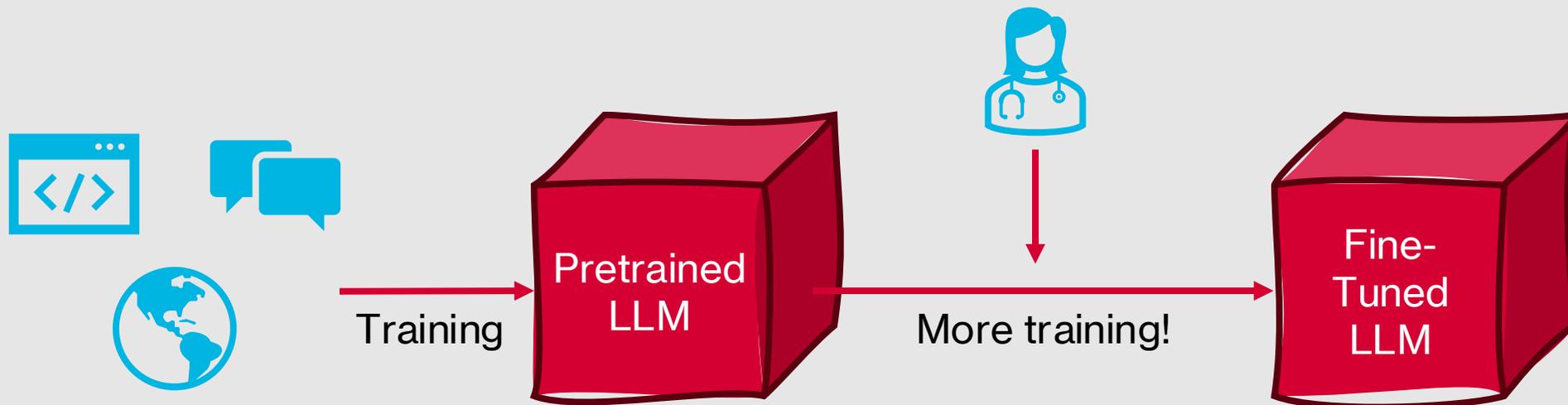
How does fine-tuning work?

- In a nutshell:
 - Take a pretrained model
 - Run additional training passes on it using new data



How does fine-tuning work?

- In a nutshell:
 - Take a pretrained model
 - Run additional training passes on it using new data



How can we fine-tune models (in more detail)?

- Several different approaches
- Popular and straightforward approach: **continued pretraining**
 - Retrain all the parameters of the model on the new data as if it just directly follows the pretraining data in the dataset
 - Same task goal (word prediction)
 - Same loss function
- This approach works well, but can be slow and expensive

How to improve efficiency when fine-tuning LLMs?



Freeze some parameters (leave them unchanged from what was learned during pretraining) and focus on training the remaining parameters

Often referred to as parameter-efficient fine-tuning (PEFT)



Use a language model as a classifier for a different task (not language modeling itself) by adding a task head

Input to task head: Top layer embedding(s) from the LLM
Output: Classifier prediction



Switch to using supervised fine-tuning

Create a dataset of prompts and their desired responses, and train the language model to predict these responses

How to improve efficiency when fine-tuning LLMs?



Freeze some parameters (leave them unchanged from what was learned during pretraining) and focus on training the remaining parameters

Often referred to as parameter-efficient fine-tuning (PEFT)



Use a language model as a classifier for a different task (not language modeling itself) by adding a task head

Input to task head: Top layer embedding(s) from the LLM
Output: Classifier prediction



Switch to using supervised fine-tuning

Create a dataset of prompts and their desired responses, and train the language model to predict these responses

Involves using the language model to produce contextual embeddings

- An important auxiliary outcome of training masked language models is that they learn **contextual embeddings** for all of the tokens in an input (including tokens that weren't masked out)
- How can we access these embeddings?
 - Use the output vector \mathbf{z}_i for a particular input token \mathbf{x}_i directly
 - Or, average across the last few layers of the model to get an averaged \mathbf{z}_i
- We can use these embeddings in all of the same types of applications where we would use other word vectors!
 - Feature representations
 - Word similarity and analogy tasks

Contextual Embeddings

Contextual Versus Static Embeddings

- Static word embeddings (e.g., Word2Vec, GloVe, or TFIDF) represent the meaning of unique vocabulary words
 - Regardless of how the vocabulary word is used, the vector will remain the same
- Contextual word embeddings represent the meaning of tokens as they appear in text
 - Vectors will only be the same if the context for two tokens is identical (very rare for this to be the case!)
 - This means that different senses of a word will be represented using different vectors (and even just the same word sense used in a slightly different context will be represented using a different vector)

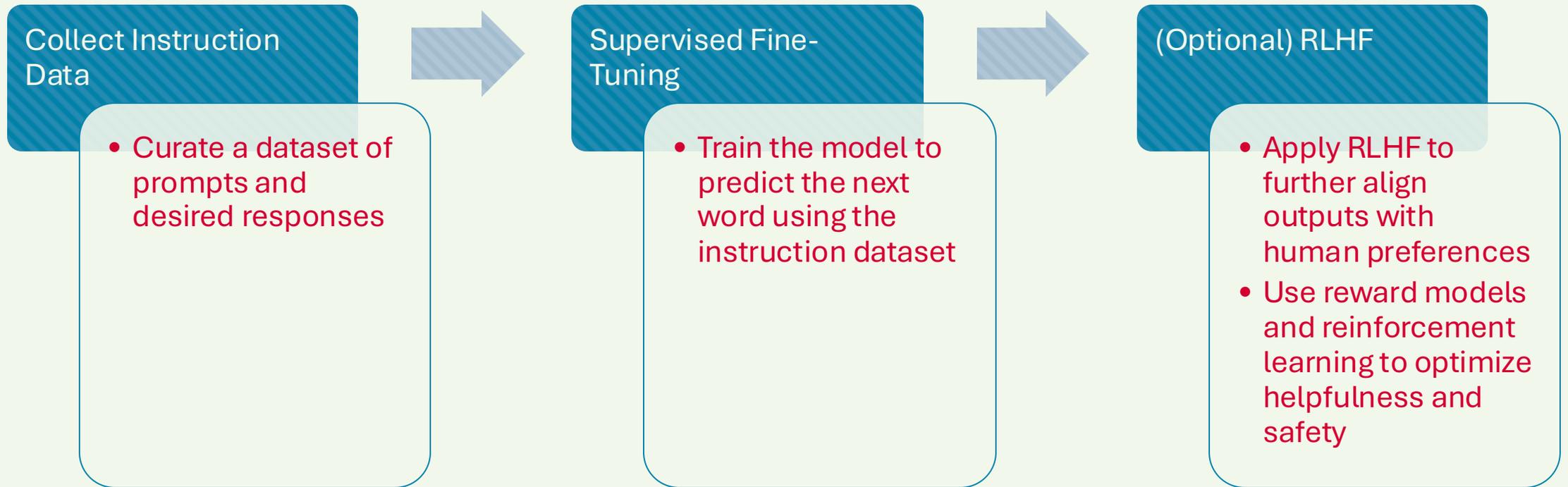
Parameter- Efficient Fine- Tuning

- Low-Rank Adaptation (LoRA) and Quantized Low-Rank Adaptation (QLoRA)
 - Inject small trainable matrices into the model
 - These matrices approximate weight updates using low-rank decomposition
 - Base model for LoRA remains in full floating point precision (typically FP16), whereas base model weights for QLoRA are 4-bit quantized
- Adapters
 - Small trainable modules inserted between layers of the base model
 - Adapters learn task-specific transformations without altering the main weights

Other Recent Fine-Tuning Strategies

- **Instruction Fine-Tuning:** Align the model with task instructions
 - Model is fine-tuned on instruction-response pairs rather than raw text
- **Reinforcement Learning from Human Feedback (RLHF):** Align the model outputs with human preferences
 - Use human-annotated preference rankings for model outputs for the same prompt to train a reward model that learns to predict these human preferences
 - Apply reinforcement learning (typically proximal policy optimization (PPO)) to generate responses, get a reward from the reward model, and update the policy to maximize the expected reward

Instruction Fine-Tuning Pipeline



Advanced Fine-Tuning Strategies

- **Mixture of Experts (MoE)** and **Mixture of Agents (MoA)**: Two related techniques to implement modular specialization
- **Memory Fine-Tuning**: Helps long-context tasks by adding a key-value memory store that persists across sessions or tasks
 - Can be combined with **retrieval-augmented generation (RAG)** for external knowledge
- **Direct Preference Optimization (DPO)**: Aligns LLMs with human preferences without reinforcement learning

Mixture of Experts and Mixture of Agents

Mixture of Experts

- Use a set of expert modules within a single large model, and a router to decide which experts to activate for a given token or input
- Experts: Each expert is a small feedforward network or specialized layer that is attuned to a particular aspect of language or domain-specific pattern
- Router: For each token, uses a prediction algorithm to determine which expert(s) should process it
 - Since only selected experts run for each token, this increases computational efficiency

Mixture of Agents

- Use multiple independent models (agents) to collaboratively solve tasks
- Agents: Each agent can be specialized for a particular subtask
- Rather than routing, agents communicate or vote on outputs using task-level orchestration frameworks or multi-agent protocols

Mixture of Experts and Mixture of Agents

Mixture of Experts

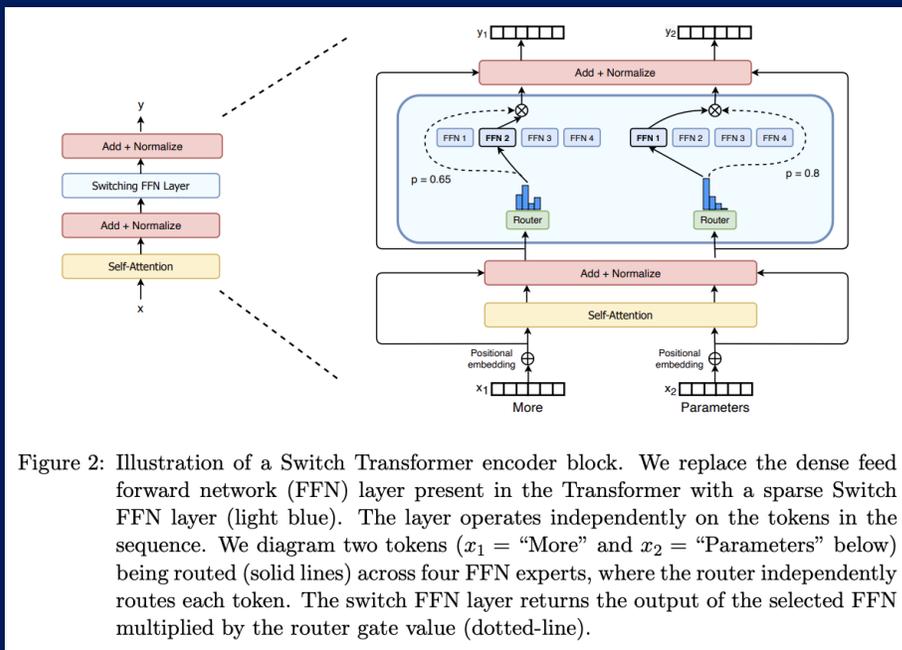


Figure 2: Illustration of a Switch Transformer encoder block. We replace the dense feed forward network (FFN) layer present in the Transformer with a sparse Switch FFN layer (light blue). The layer operates independently on the tokens in the sequence. We diagram two tokens (x_1 = "More" and x_2 = "Parameters" below) being routed (solid lines) across four FFN experts, where the router independently routes each token. The switch FFN layer returns the output of the selected FFN multiplied by the router gate value (dotted-line).

William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: scaling to trillion parameter models with simple and efficient sparsity. J. Mach. Learn. Res. 23, 1, Article 120 (January 2022), 39 pages.

Mixture of Agents

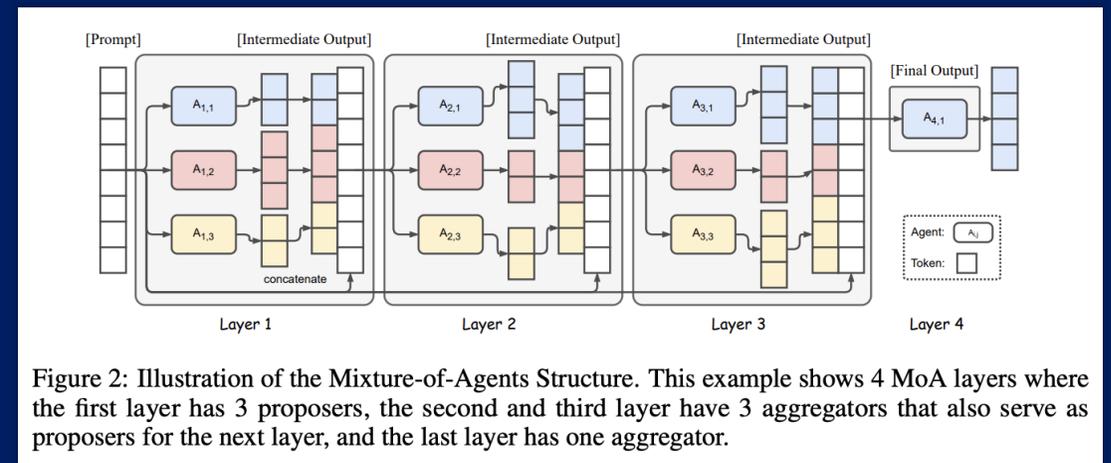


Figure 2: Illustration of the Mixture-of-Agents Structure. This example shows 4 MoA layers where the first layer has 3 proposers, the second and third layer have 3 aggregators that also serve as proposers for the next layer, and the last layer has one aggregator.

Wang, J., Jue, W.A.N.G., Athiwaratkun, B., Zhang, C. and Zou, J., 2025. Mixture-of-Agents Enhances Large Language Model Capabilities. In *The Thirteenth International Conference on Learning Representations*.

Direct Preference Optimization

Collect preference data, similarly to RLHF

- Humans compare two candidate responses for the same prompt and indicate which one is better

Implement a pairwise loss function with a contrastive objective

- For each pair (preferred versus non-preferred response for the same prompt), increase the probability associated with the preferred response and decrease the probability associated with the non-preferred one)

Fine-tune the LLM using this pairwise loss function across many preference pairs

Generative AI in an NLP context refers to **LLMs**: large Transformer-based language models

Many LLMs are pretrained using **teacher forcing**, which forces the model to learn to predict the next word based on the correct input context

LLMs are generally trained using large web corpora, which should be filtered for quality and safety purposes

Many different **prompting** techniques can be used to harness predictive and generalization capacities of LLMs

If we don't want to use a pretrained LLM directly for our task, we can also **fine-tune** it to perform better in our desired task setting

Summary:
LLMs

This Week's Topics

Large Language Models
Prompting
Fine-Tuning

Thursday

Tuesday



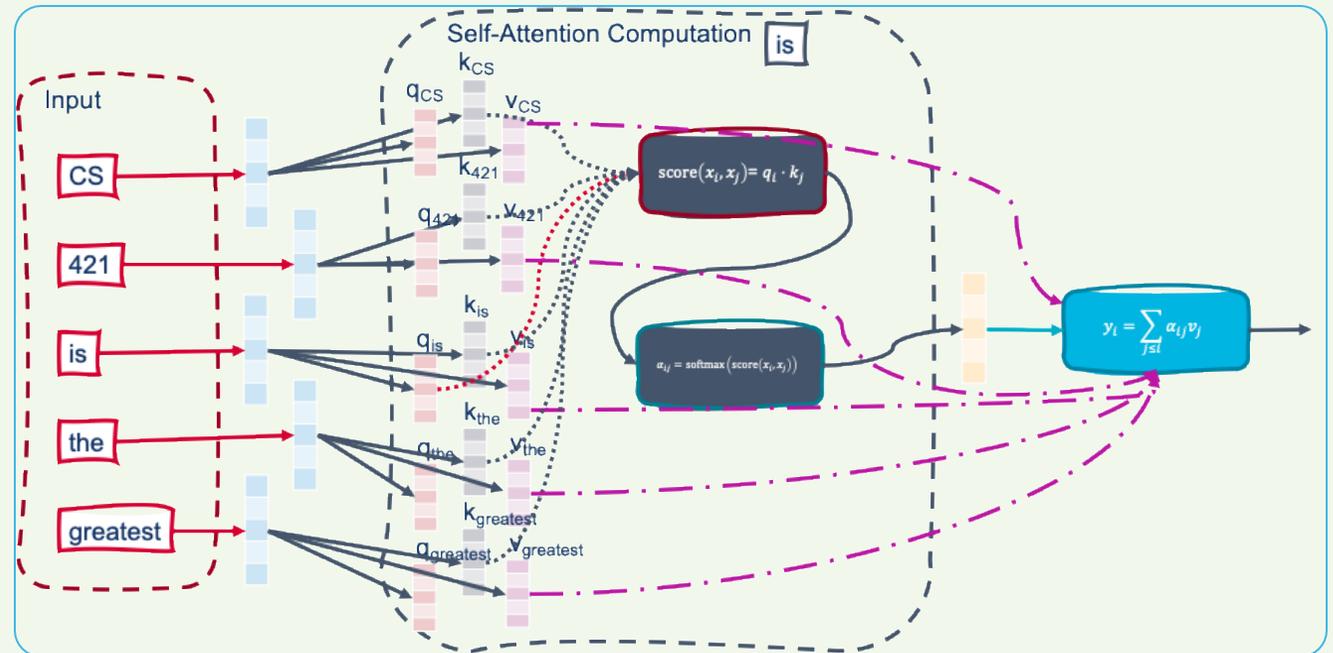
Masked Language Modeling
Popular Bidirectional
Transformer Architectures
More Fine-Tuning
Best Practices for Data-
Driven Learning

LLMs Beyond GPT

- GPT-powered models are the most popular LLMs for consumer-facing chat applications, but they are far from the only type of LLM used by NLP researchers!
- Recall: With other sequence processing architectures (e.g., RNNs), **bidirectionality** is known to improve performance

Can Transformers be bidirectional?

- No theoretical reason why not!
- In causal LLMs, we artificially constrain the model to consider only previous history because considering future context would trivialize the task of next word prediction



- In general, **sequence labeling** and **sequence classification** tasks (not generation!)
 - Tasks where you can reasonably assume you'll have the full context for all unit(s) to be labeled within a text sequence, and that no additional context will later be added
- Examples:
 - Classifying social media posts from a dataset
 - Assigning grammatical labels to words within an input sentence

In what settings would bidirectional Transformers be useful?

Bidirectional Transformers

- Useful?
- However, implementing bidirectional Transformers requires that we rethink some aspects of our LLM
 - **New training objective needed** (next word prediction makes no sense with a model that can view the next word)
 - **New inference approaches needed** (without next word prediction, the model is unlikely to generate text sequences of a similar quality to those produced using causal LLMs)

What training objective works well for a bidirectional Transformer?

What we know from causal LLMs:

- Self-supervised learning allows us to train on lots of data without requiring lots of manual annotation
- Next word prediction works well at helping LLMs understand relationships between real-world concepts

The goal, then:

- Come up with training objectives that capture the same type of information as next word prediction, but that don't rely on predicting a fixed-position future word

The solution:

- **Masked language modeling**

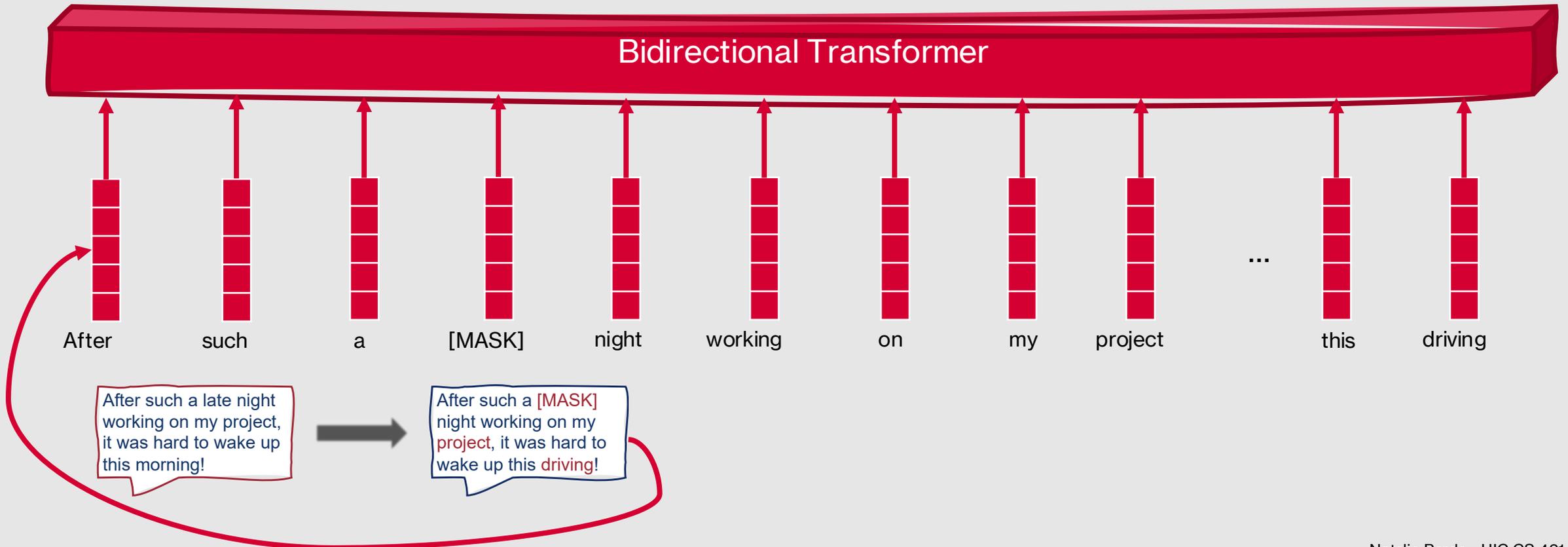
Masked Language Modeling

- Randomly select a subset of tokens from the training input and:
 - Replace some of them with [MASK] tokens
 - Replace some of them with other randomly sampled tokens
 - Leave some of them unchanged
- For each sampled token, try to predict what the correct token is
- The more general form of this task is often referred to as **denoising**: in essence, you add noise to an input by corrupting it in some way, and then try to remove the noise as a learning objective

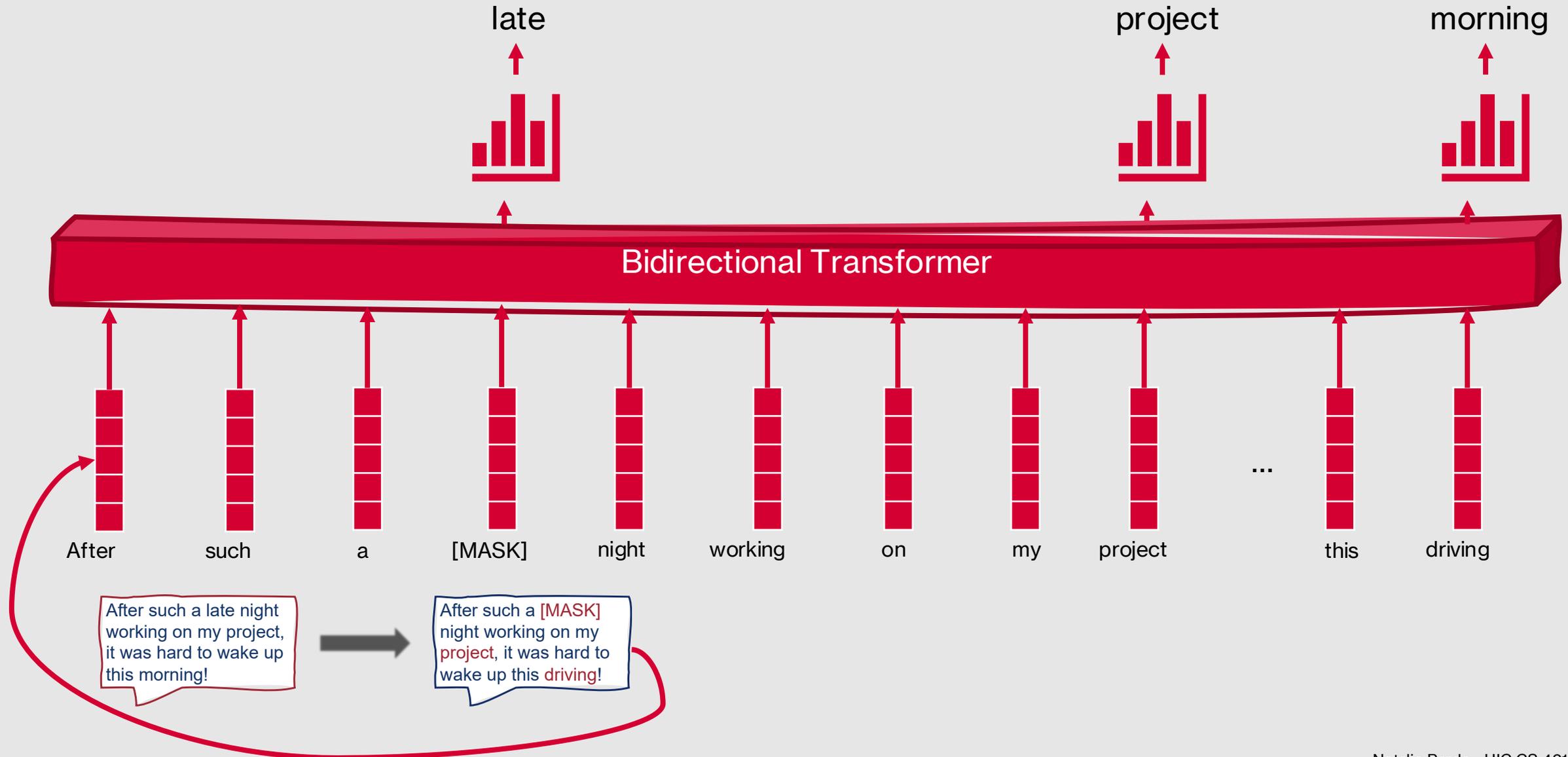
Masked Language Modeling



Masked Language Modeling



Masked Language Modeling

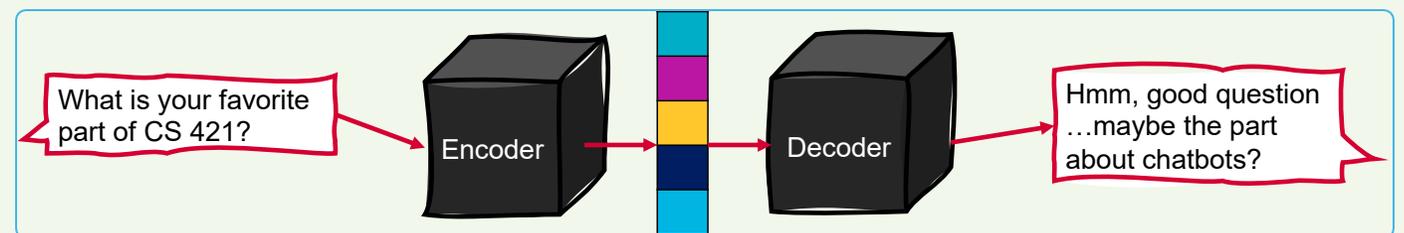


Key Differences between Masked Language Modeling and Next Word Prediction

- Requires mapping an input sequence to an output sequence of the same length
 - Masked tokens may be anywhere within the input sequence!
- In doing so, emphasizes the learning of high-quality **contextual representations** of all input tokens
- These contextual representations can be broadly useful across many applications

Decoder-Only Versus Encoder- Only Models

- Recall (from week one!) our discussion of encoder-decoder models
- Causal LLMs like GPT are often referred to as **decoder-only** models since they focus on generating output
- With this analogy, bidirectional LLMs could be thought of as **encoder-only models**: the focus is on generating high-quality contextual encodings of the input, rather than on producing natural output



This Week's Topics

Large Language Models
Prompting
Fine-Tuning

Thursday

Tuesday



Masked Language Modeling
Popular Bidirectional
Transformer Architectures
More Fine-Tuning
Best Practices for Data-
Driven Learning

Bidirectional Transformer Architecture

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin Ming-Wei Chang Kenton Lee Kristina Toutanova
Google AI Language
{jacobdevlin, mingweichang, kentonl, kristout}@google.com

Abstract

We introduce a new language representation model called BERT, which stands for Bidirectional Encoder Representations from Transformers. Unlike recent language representation models (Peters et al., 2018a; Radford et al., 2018), BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications. BERT is conceptually simple and empirically powerful. It obtains new state-of-the-art results on eleven natural language processing tasks, including pushing the GLUE score to 80.5% (7.7% point absolute improvement), MultiNLI accuracy to 86.7% (4.6% absolute improvement), SQuAD v1.1 question answering Test F1 to 93.2 (1.5 point absolute improvement) and SQuAD v2.0 Test F1 to 83.1 (5.1 point absolute improvement).

1 Introduction

Language model pre-training has been shown to be effective for improving many natural language processing tasks (Dai and Le, 2015; Peters et al., 2018; Radford et al., 2018; Howard and Ruder, 2018). These include sentence-level tasks such as natural language inference (Bowman et al., 2015; Williams et al., 2018) and paraphrasing (Dolan and Brockett, 2005), which aim to predict the relationships between sentences by analyzing them holistically, as well as token-level tasks such as named entity recognition and question answering, where models are required to produce fine-grained output at the token level (Tjong Kim Sang and De Meulder, 2003; Rajpurkar et al., 2016).

There are two existing strategies for applying pre-trained language representations to downstream tasks: *feature-based* and *fine-tuning*. The feature-based approach, such as ELMo (Peters et al., 2018a), uses task-specific architectures that include the pre-trained representations as additional features. The fine-tuning approach, such as the Generative Pre-trained Transformer (OpenAI GPT) (Radford et al., 2018), introduces minimal task-specific parameters, and is trained on the downstream tasks by simply fine-tuning *all* pre-trained parameters. The two approaches share the same objective function during pre-training, where they use unidirectional language models to learn general language representations.

We argue that current techniques restrict the power of the pre-trained representations, especially for the fine-tuning approaches. The major limitation is that standard language models are unidirectional, and this limits the choice of architectures that can be used during pre-training. For example, in OpenAI GPT, the authors use a left-to-right architecture, where every token can only attend to previous tokens in the self-attention layers of the Transformer (Vaswani et al., 2017). Such restrictions are sub-optimal for sentence-level tasks, and could be very harmful when applying fine-tuning based approaches to token-level tasks such as question answering, where it is crucial to incorporate context from both directions.

In this paper, we improve the fine-tuning based approaches by proposing BERT: Bidirectional Encoder Representations from Transformers. BERT alleviates the previously mentioned unidirectionality constraint by using a “masked language model” (MLM) pre-training objective, inspired by the Cloze task (Taylor, 1953). The masked language model randomly masks some of the tokens from the input, and the objective is to predict the original vocabulary id of the masked

4171

Proceedings of NAACL-HLT 2019, pages 4171–4186
Minneapolis, Minnesota, June 2–June 7, 2019. ©2019 Association for Computational Linguistics

- Aside from the pretraining objective, are any major changes required from the Transformer architectures used for causal LLMs?
 - Nope!
 - Just allow the model to access the full context and change the pretraining objective
 - Different types of bidirectional Transformers can be created, just like different types of causal Transformers can be created, by stacking Transformer blocks and using variations of standard pretraining tasks
- Extremely popular bidirectional Transformer architecture: **BERT**
 - Paper Link: <https://aclanthology.org/N19-1423.pdf>

Bidirectional Encoder Representations from Transformers (BERT)

- Commonly accepted as a standard bidirectional Transformer **benchmark** model
- Implemented using:
 - 12 Transformer blocks, each of which have 12 bidirectional attention heads
 - 768-dimensional hidden layers
 - A subword vocabulary of 30,000 tokens
 - A fixed input length of 512 subword tokens

Many variations and extensions of BERT have been developed and are very popular among NLP researchers.

- **RoBERTa:** Trains for longer and removes a secondary pretraining task
- **XLNet:** Pretrains similarly to RoBERTa but with multiple languages in the input dataset
- **SpanBERT:** Focuses on spans of text
- **DistilBERT:** Distills BERT such that fewer parameters are needed (more efficient)

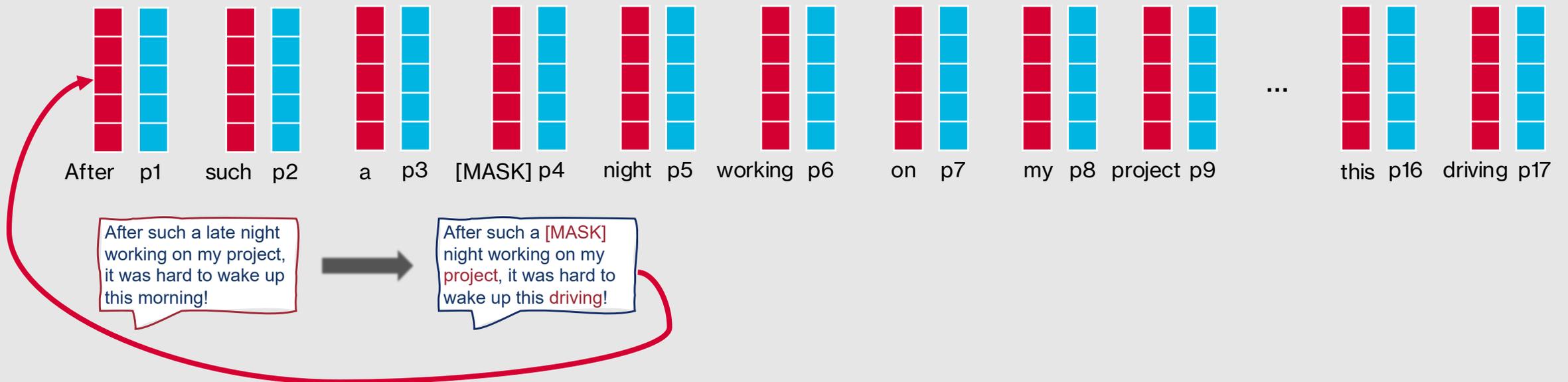
- Multilingual subword vocabulary with 250,000 tokens
- 24 Transformer blocks, each with 16 attention heads
- 1024-dimensional hidden layers
- A fixed input length of 512 subword tokens
- This amounts to 550,000,000 trainable parameters overall
- Note that this is still relatively small compared to state-of-the-art causal LLMs (Llama 3 has 405,000,000,000 parameters)!
 - Masked language models tend to be much smaller, and thus more efficient to train, than causal LLMs

XLNet- RoBERTa's Architecture

BERT is trained to perform two tasks.

- **Masked language modeling**

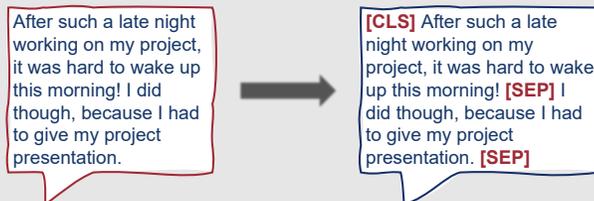
- As described earlier, but include position embeddings
- In BERT, 15% of tokens per sequence are sampled for learning (80% of these are replaced with [MASK], 10% are replaced with other randomly sampled tokens, and 10% are left unchanged)



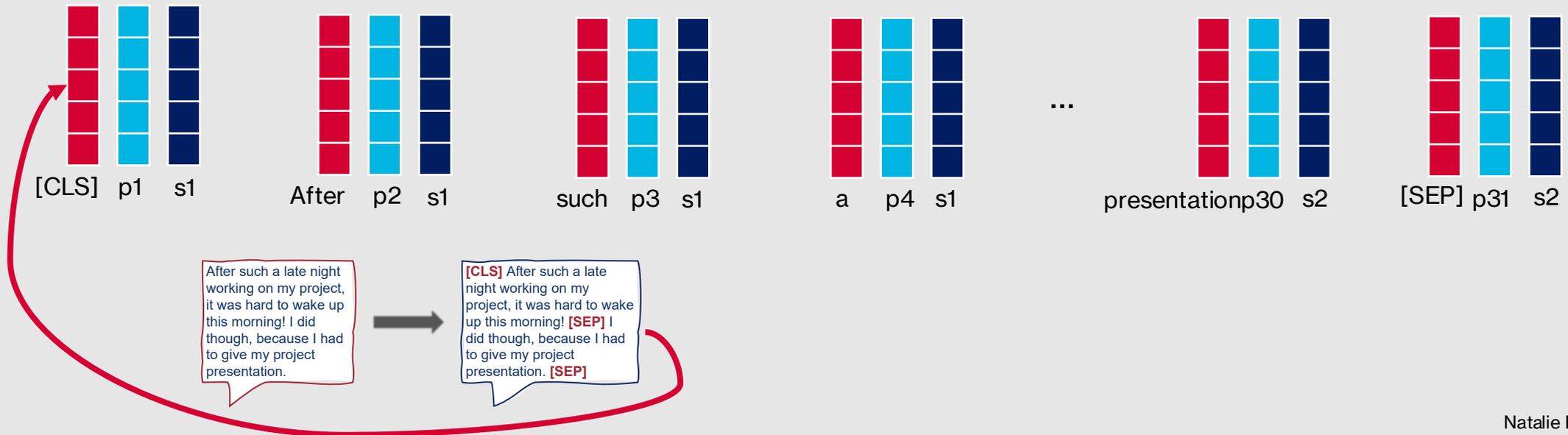
BERT is trained to perform two tasks.

- Masked language modeling
 - As described earlier, but include position embeddings
 - In BERT, 15% of tokens per sequence are sampled for learning (80% of these are replaced with [MASK], 10% are replaced with other randomly sampled tokens, and 10% are left unchanged)
- **Next sentence prediction**
 - Predict whether pairs of sentences are actually adjacent to one another in text
 - Prepend a [CLS] token to the pair of sentences
 - Separate the two sentences using a [SEP] token
 - Add segment embeddings to the model
 - Assign a label based on the representation learned for the [CLS] token

Next Sentence Prediction

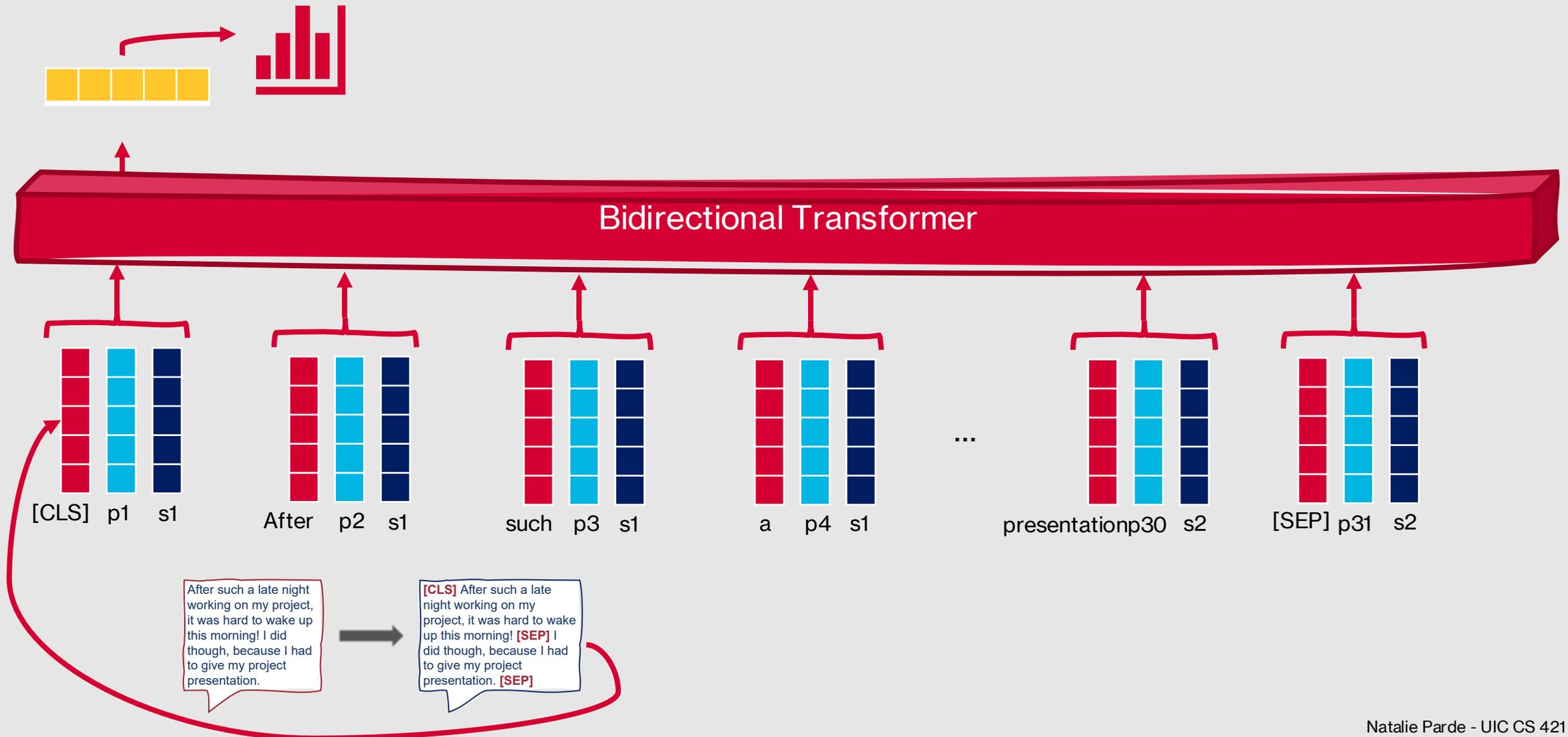


Next Sentence Prediction



Next Sentence Prediction

1 (second sentence does follow the first)



Why add next sentence prediction?

- For many tasks, it is beneficial to understand the relationships between pairs of sentences
 - Paraphrase detection
 - Entailment
 - Discourse coherence
- This more global type of understanding isn't directly encoded using masked language modeling
- Next sentence prediction allows us to target this more directly while still using self-supervised learning
 - 50% of training pairs are taken from the corpus directly
 - 50% are created by randomly sampling a second sentence from elsewhere in the corpus

This Week's Topics

Large Language Models
Prompting
Fine-Tuning

Thursday

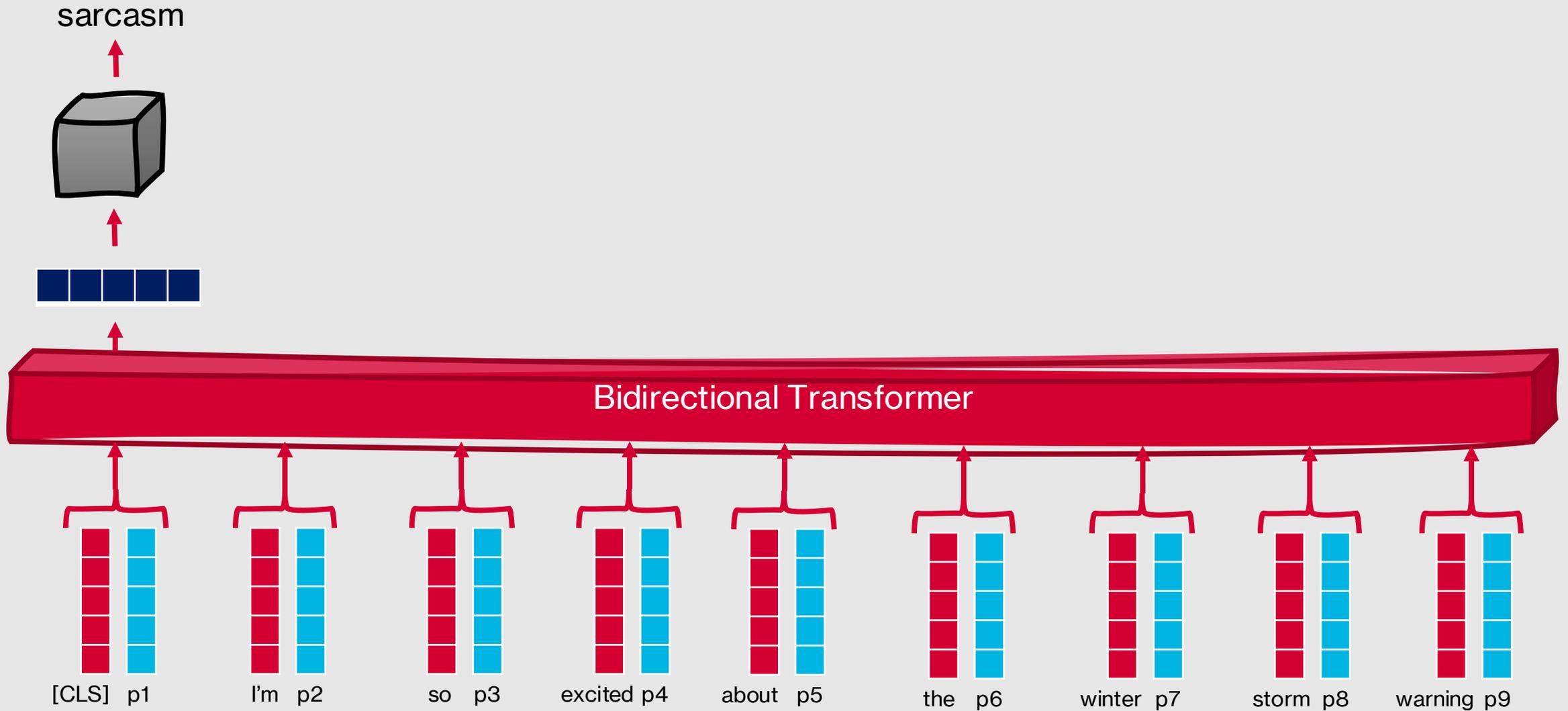
Tuesday

Masked Language Modeling
Popular Bidirectional
Transformer Architectures
 More Fine-Tuning
Best Practices for Data-Driven Learning

How do we finetune bidirectional Transformers?

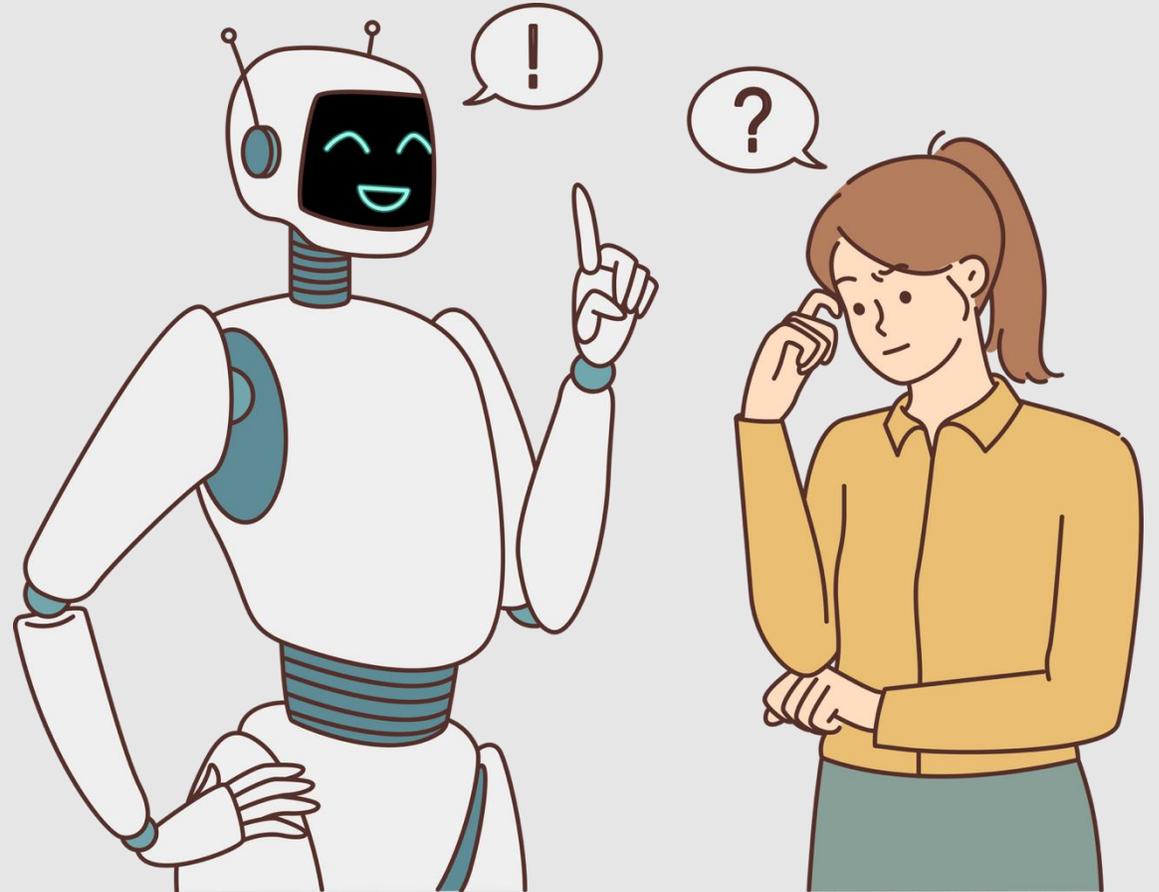
- Take a large model that has already been trained for some other task
 - Example: BERT has already been trained for masked language modeling and next sentence prediction
- Add a task head to the model
 - Task-specific layer(s) that take the input representations from the pretrained model and produce your desired output
- Update the parameters for the task head while ignoring or only minimally adjusting the weights for the pretrained model
 - This will require that you have supervised training data for your target task

Example: Finetuned Sarcasm Detector



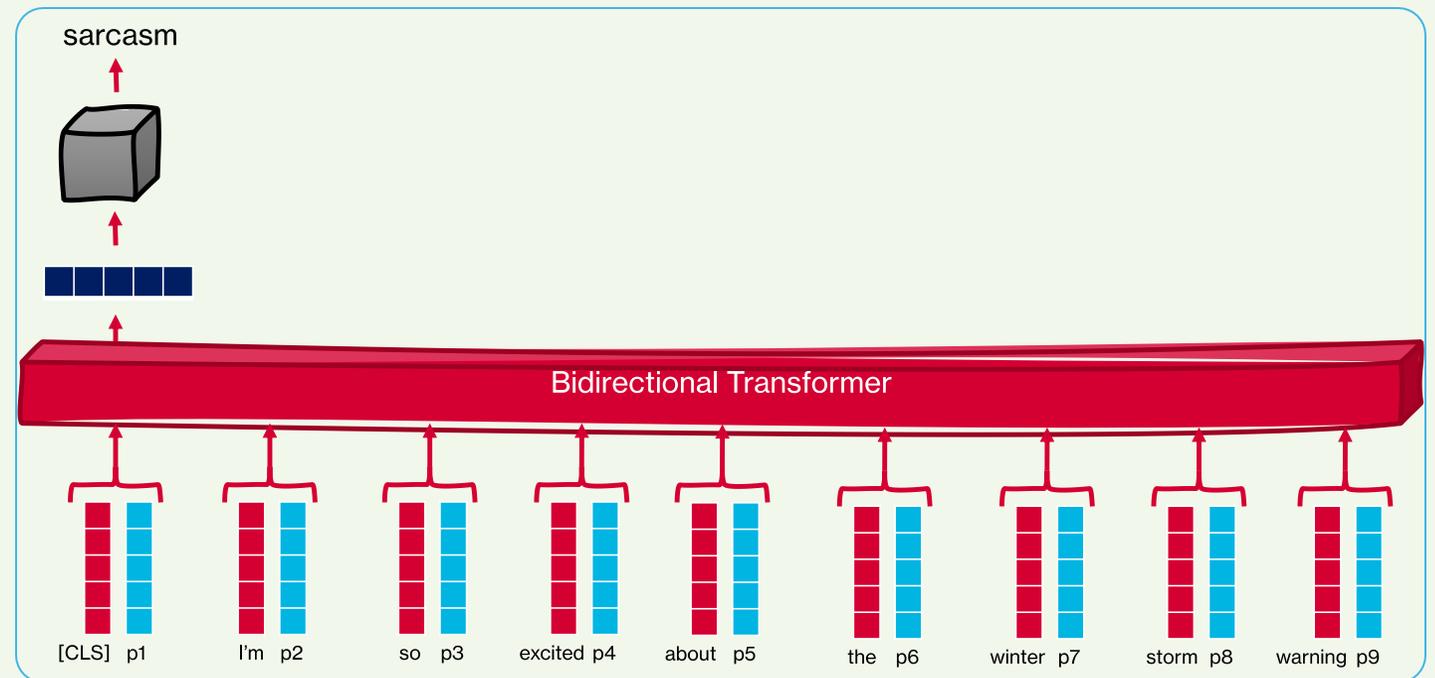
Why does this work?

- Pretraining on large datasets allows language models to build high-quality representations facilitating **general language understanding**
- In many cases, this knowledge can be **reused** across many tasks
 - Sentences are likely to have similar structure across many language domains
 - Common sense knowledge is likely to transfer across problem settings
 - Semantic relationships often hold across tasks
- Specialized tasks often have much **less data available** than the tasks used to train large language models
- By finetuning an encoder-based LLM to perform the specialized task, we can retain the useful general language information we've learned and use it to help us more **efficiently and effectively** solve our task



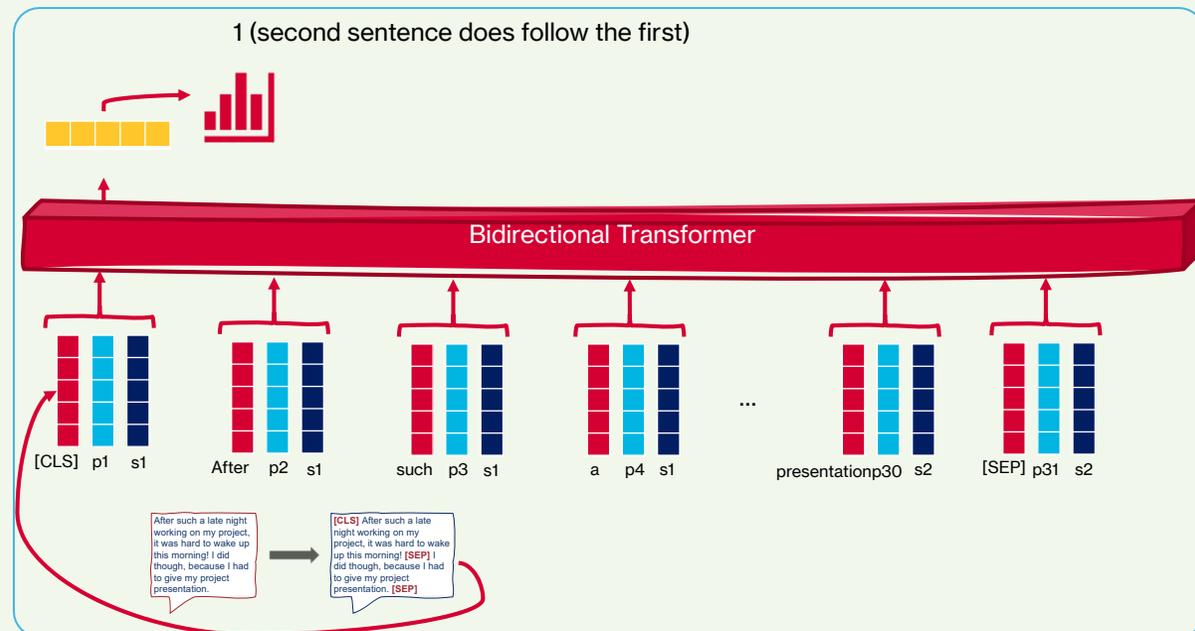
We can use finetuning in many different task setups.

- Sequence classification
 - Add a unique [CLS] token to the beginning of the input, and use the output vector for that token as input to the task head



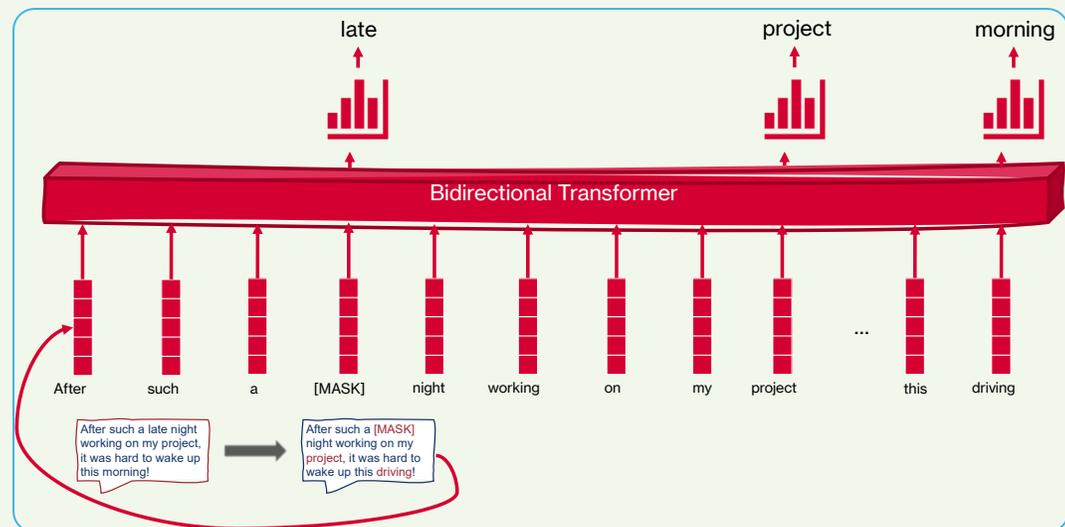
We can use finetuning in many different task setups.

- Sequence classification
 - Add a unique [CLS] token to the beginning of the input, and use the output vector for that token as input to the task head
- Sequence pair classification
 - Set the task up similarly to the next sentence prediction pretraining objective (add a [CLS] token and also separate sentences using [SEP] tokens)



We can use finetuning in many different task setups.

- Sequence classification
 - Add a unique [CLS] token to the beginning of the input, and use the output vector for that token as input to the task head
- Sequence pair classification
 - Set the task up similarly to the next sentence prediction pretraining objective (add a [CLS] token and also separate sentences using [SEP] tokens)
- Sequence labeling
 - Set the task up similarly to the masked language modeling task, but predict output labels for every token in the input (not just those that are masked)



This Week's Topics

Large Language Models
Prompting
Fine-Tuning

Thursday

Tuesday

Masked Language Modeling
Popular Bidirectional
Transformer Architectures
More Fine-Tuning
Best Practices for Data-
Driven Learning



Evaluating LLMs

- **Perplexity** (introduced a few weeks ago with n-gram language modeling) is also used to evaluate LLMs
 - *However*, keep in mind: With LLMs, perplexity will be sensitive to differences in the tokenization algorithm
 - Make sure that if you're comparing LLMs using perplexity, they were trained using the same tokenizer!
- **Task-specific metrics**
- **Power metrics**
 - Model size
 - Training and inference times
 - Energy usage
- **Fairness**

We can also use contextual embeddings resulting from LLMs for more general-purpose similarity metrics.

- BERTScore

- Get contextual embeddings for tokens in the candidate and reference
- Find the cosine similarity between each possible pair of (candidate, reference) tokens
- Match each token in the candidate with its most similar token in the reference
- Use the similarity values for those matches to calculate BERT-precision and BERT-recall

- $R_{\text{BERT}} = \frac{1}{|x|} \sum_{x_i \in x} \max_{\hat{x}_j \in \hat{x}} \mathbf{x}_i^T \hat{\mathbf{x}}_j$

- $P_{\text{BERT}} = \frac{1}{|\hat{x}|} \sum_{\hat{x}_j \in \hat{x}} \max_{x_i \in x} \mathbf{x}_i^T \hat{\mathbf{x}}_j$

- Use BERT-precision and BERT-recall to compute F_1 (same equation as usual)

Measuring Similarity between Contextual Embeddings

- Contextual embeddings tend to be more similar across all words than static embeddings
- This means that the cosine similarity for most contextual vectors will be close to 1.0
- If we're interested in measuring vector similarity between tokens, we would ideally have more **isotropic** vectors, such that the expected cosine similarity between a randomly sampled pair of embeddings would be 0

- Standardize the vectors by subtracting the mean vector and dividing by the standard deviation
- Mean vector of all embeddings in a corpus C:
 - $\mu = \frac{1}{|C|} \sum_{\mathbf{x} \in C} \mathbf{x}$
- Standard deviation:
 - $\sigma = \sqrt{\frac{1}{|C|} \sum_{\mathbf{x} \in C} (\mathbf{x} - \mu)^2}$
- Standardized vector:
 - $\mathbf{z} = \frac{\mathbf{x} - \mu}{\sigma}$

**How can we
make our
vectors more
isotropic?**

Potential Harms from LLMs

Hallucination: LLMs are trained to generate tokens that would predictably come next in a coherent output, but most training algorithms do *not* have a way to enforce correctness

Toxic Language: Just like in earlier data-driven models (e.g., GloVe vectors), LLMs have been shown to replicate human stereotypes and negative attitudes about demographic groups

Cultural Bias: Web-based training data is disproportionately authored by people living in western countries, and as a result, LLMs often take a western cultural perspective

Information Leakage: If sensitive information isn't properly filtered from the training data, adversaries can extract it using malicious prompts

Misinformation: Since LLMs effectively generate coherent language, they can be used by malicious actors to generate harmful but convincing misinformation or propaganda

Finding ways to address these potential harms is an important and active research area in NLP!

Which model should you use?

- Many approaches are now available for us to use when developing NLP systems, ranging from early rule-based techniques to very recent prompt-based methods
- In general, with each new modeling era of NLP we have sacrificed some degree of control and interpretability for increased performance

Rule-Based

- Complete control and interpretability, but very limited ability to generalize

Statistical

- We have immediate access to feature values and weights and can generalize a bit more broadly, but we require supervised training data

Neural End-to-End

- We no longer know our feature values or weights, but we can generalize more broadly and we know our exact inputs and outputs

Pretrain and Finetune

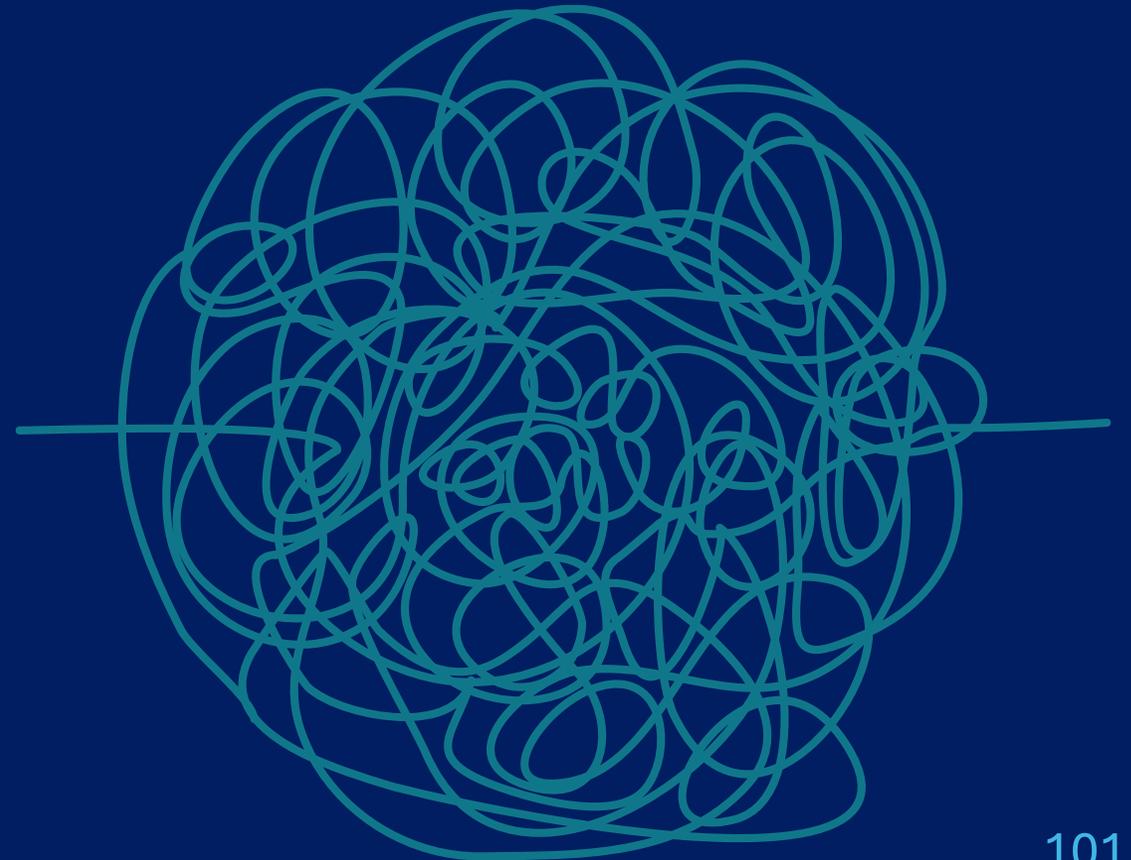
- We are generalizing from a wealth of broad knowledge, although we only know specific data/task details pertaining to our target task

Pretrain and Prompt

- We don't know exactly how or why our model is making its decisions, but we achieve strong performance and no longer require supervised training data

Remember, deep learning isn't necessarily the best solution in all scenarios!

- Less interpretable
 - Particularly important to consider when dealing with sensitive tasks (e.g., classifying health-related documents)
 - Prompt-based approaches may generate inaccurate output and present it confidently
- May overfit with very low-resource problems
- May overcomplicate the solution
 - In some cases, a naïve Bayes model may work just as well as a complex deep learning approach



Tools for Implementing Generative AI Systems

- Pretrained Language Models
 - HuggingFace Model Hub: <https://huggingface.co/models>
- Deep Learning Frameworks
 - PyTorch: <https://pytorch.org/>
 - TensorFlow: <https://www.tensorflow.org/>
- Prompt Tuning Frameworks
 - OpenPrompt: <https://github.com/thunlp/OpenPrompt>

Other Practical Guidelines for Data-Driven NLP

- When using publicly available models or releasing your own model or data to the public, it is good to be transparent about:
 - How the data was collected and/or the model was trained
 - Whether your resource is open or closed
 - Other information applicable to the greater community

Datasheets

- In electronics, new applications typically come with datasheets that describe important information for understanding how to use the application properly
- AI research until recently has had no such equivalent, but this is quickly changing!
- It is now considered a best practice to release a **datasheet** with every new dataset
 - Paper introducing datasheets: <https://www.microsoft.com/en-us/research/uploads/prod/2019/01/1803.09010.pdf>

Movie Review Polarity

Thumbs Up? Sentiment Classification using Machine Learning Techniques

Motivation for Dataset Creation

Why was the dataset created? (e.g., were there specific tasks in mind, or a specific gap that needed to be filled?)

The dataset was created to enable research on predicting sentiment polarity: given a piece of (English) text, predict whether it has a positive or negative affect or stance toward its topic. It was created intentionally with that task in mind, focusing on movie reviews as a place where affect/sentiment is frequently expressed.¹

What (other) tasks could the dataset be used for? Are there obvious tasks for which it should *not* be used?

The dataset could be used for anything related to modeling or understanding movie reviews. For instance, one may induce a lexicon of words/phrases that are highly indicative of sentiment polarity, or learn to automatically generate movie reviews.

Has the dataset been used for any tasks already? If so, where are the results so others can compare (e.g., links to published papers)?

At the time of publication, only the original paper <http://xxx.lanl.gov/pdf/cs/0409058v1>. Between then and 2012, a collection of papers that used this dataset was maintained at <http://www.cs.cornell.edu/people/pabo/movie%2Dreview%2Ddata/otherexperiments.html>.

Who funded the creation of the dataset? If there is an associated grant, provide the grant number.

Funding was provided through five distinct sources: the National Science Foundation, the Department of the Interior, the National Business Center, Cornell University, and the Sloan Foundation.

Dataset Composition

What are the instances? (that is, examples; e.g., documents, images, people, countries) Are there multiple types of instances? (e.g., movies, users, ratings; people, interactions between them; nodes, edges)

The instances are movie reviews extracted from newsgroup postings, together with a sentiment rating for whether the text corresponds to a review with a rating that is either strongly positive (high number of stars) or strongly negative (low number of stars). The polarity rating is binary {positive,negative}. An example instance is shown in Figure 1.

Are relationships between instances made explicit in the data (e.g., social network links, user/movie ratings, etc.)?

None explicitly, though the original newsgroup postings include poster name and email address, so some information could be extracted if needed.

How many instances of each type are there?

There are 1400 instances in total in the original (v1.x versions) and 2000 instances in total in v2.0 (from 2014).

¹Information in this datasheet is taken from one of five sources; any errors that were introduced are our fault. <http://www.cs.cornell.edu/people/pabo/movie-review-data/>; <http://xxx.lanl.gov/pdf/cs/0409058v1>; <http://www.cs.cornell.edu/people/pabo/movie-review-data/polaritydata>; README.1.0.txt; <http://www.cs.cornell.edu/people/pabo/movie-review-data/poldata>; README.2.0.txt.

these are words that could be used to describe the emotions of john sayles' characters in his latest , limbo . but no , i use them to describe myself after sitting through his latest little exercise in indie egomania . i can forgive many things . but using some hackneyed , whacked-out , screwed-up * non * , ending on a movie is unforgivable . i walked a half-mile in the rain and sat through two hours of typical , plodding sayles melodrama to get cheated by a complete and total copout finale . does sayles think he's roger corman ?

Figure 1. An example "negative polarity" instance, taken from the file `neg/cv452.tok-18656.txt`.

What data does each instance consist of? "Raw" data (e.g., unprocessed text or images)? Features/attributes? Is there a label/target associated with instances? If the instances are related to people, are subpopulations identified (e.g., by age, gender, etc.) and what is their distribution? Each instance consists of the text associated with the review, with obvious ratings information removed from that text (some errors were found and alter fixed). The text was down-cased and HTML tags were removed. Boilerplate newsgroup header/footer text was removed. Some additional unspecified automatic filtering was done. Each instance also has an associated target value: a positive (+1) or negative (-1) rating based on the number of stars that that review gave (details on the mapping from number of stars to polarity is given below in "Data Preprocessing").

Is everything included or does the data rely on external resources? (e.g., websites, tweets, datasets) If external resources, a) are there guarantees that they will exist, and remain constant, over time; b) is there an official archival version. Are there licenses, fees or rights associated with any of the data? Everything is included.

Are there recommended data splits or evaluation measures? (e.g., training, development, testing, accuracy/AUC)

The instances come with a "cross-validation tag" to enable replication of cross-validation experiments; results are measured in classification accuracy.

What experiments were initially run on this dataset? Have a summary of those results and, if available, provide the link to a paper with more information here.

Several experiments are reported in the README for baselines on this data, both on the original dataset (Table 1) and the cleaned version (Table 2). In these results, NB=Naive Bayes, ME=Maximum Entropy and SVM=Support Vector Machine. The feature sets include unigrams (with and without counts), bigrams, part of speech features, and adjectives-only.

Data Collection Process

How was the data collected? (e.g., hardware apparatus/sensor, manual human curation, software program, software interface/API; how were these constructs/measures/methods validated?)

The data was collected by downloading reviews from the IMDb archive of the `rec.arts.movies.reviews` newsgroup, at <http://reviews.imdb.com/Reviews>.

Who was involved in the data collection process? (e.g., students, crowdworkers) How were they compensated? (e.g., how much were crowdworkers paid?)

What should be included in a datasheet?

- **Motivation for dataset creation**
- Dataset composition
- Data collection process
- Data preprocessing
- Dataset distribution
- Dataset maintenance
- Legal and ethical considerations

Why create this dataset in the first place (and who funded it)?

What should be included in a datasheet?

- Motivation for dataset creation
- **Dataset composition**
- Data collection process
- Data preprocessing
- Dataset distribution
- Dataset maintenance
- Legal and ethical considerations

Why create this dataset in the first place?

What are the instances in the data and the relationships between them? How large (and comprehensive) is the dataset?

What should be included in a datasheet?

- Motivation for dataset creation
- Dataset composition
- **Data collection process**
- Data preprocessing
- Dataset distribution
- Dataset maintenance
- Legal and ethical considerations

Why create this dataset in the first place?

What are the instances in the dataset?

be
cc

How was the data collected, who was involved, and how long did it take? (Is anything redacted from the data?)

What should be included in a datasheet?

- Motivation for dataset creation
- Dataset composition
- Data collection process
- **Data preprocessing**
- Dataset distribution
- Dataset maintenance
- Legal and ethical considerations

Why create this dataset in the first place?

What are the instances in the dataset?

How was the data collected, who was involved?

How was the data collected, who was involved, how long did it take? (Is the data anonymized/redacted from the data?)

How was the data preprocessed (and is the raw data still available)?

What should be included in a datasheet?

- Motivation for dataset creation
- Dataset composition
- Data collection process
- Data preprocessing
- **Dataset distribution**
- Dataset maintenance
- Legal and ethical considerations

Why create this dataset in the first place? What are the instances in the dataset and the variables?

What are the instances in the dataset and the variables?

How was the data collected, who was involved, and how long did it take? (Is the data representative?)

How was the data collected, who was involved, and how long did it take? (Is the data representative?)

What license is the data distributed under? (Are there any fees or access/export restrictions?)

What should be included in a datasheet?

- Motivation for dataset creation
- Dataset composition
- Data collection process
- Data preprocessing
- Dataset distribution
- **Dataset maintenance**
- Legal and ethical considerations

Why create this dataset in the first place?

What are the instances in the dataset?

How was the data collected, who was involved?

How was the data collected, who was involved?

How long did it take? (Is the data still being collected?)

What license is the data distributed under? (Can others use the data?)

Who will support the dataset, and for how long? (Will the dataset be extended, and can others extend it?)

What should be included in a datasheet?

- Motivation for dataset creation
- Dataset composition
- Data collection process
- Data preprocessing
- Dataset distribution
- Dataset maintenance
- **Legal and ethical considerations**

Why create this dataset in the first place?

What are the instances in the dataset?

How was the data collected, who was involved?

How long did it take? (Is it a large dataset?)

What license is the data distributed under? (Is it open data?)

Who will support the dataset, and for how long?

Were the people represented by the dataset informed of the data's collection, and was the data collection reviewed by an ethical board? (Does the dataset include information that might be sensitive, confidential, inappropriate, or offensive?)

What about datasheets for trained models?

- Less popular so far, but there is also increasing interest in this
- Many NLP research venues now require general-purpose “responsible AI” checklists that cover both data collection and model development
 - Responsible NLP Research Checklist: <https://aclrollingreview.org/responsible-NLPresearch/>
 - Checklist for responsible data use in NLP: <https://aclanthology.org/2021.findings-emnlp.414.pdf>
 - Reproducibility in NLP checklist: <https://aclanthology.org/D19-1224.pdf>

Increasing compute requirements have recently opened new questions about data and model access.

- Until recently, most generally-useful NLP models could be reasonably expected to run on standard consumer-grade or academic hardware
 - Naïve Bayes and logistic regression are lightweight models that can be trained on a laptop!
- However, high-performing general-purpose models today tend to be LLMs with massive compute requirements

[NousResearch/Meta-Llama-3-8B-Instruct](#) (7.6B)

int4 int8 float16/bfloat16 float32

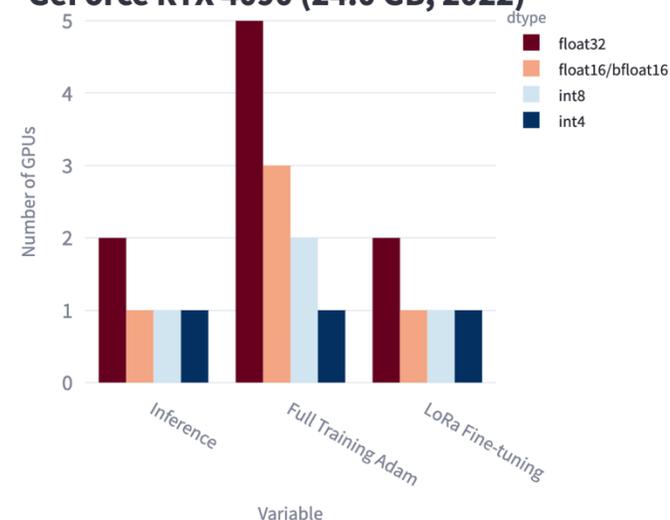
⚠ You require 2 GPUs for Inference

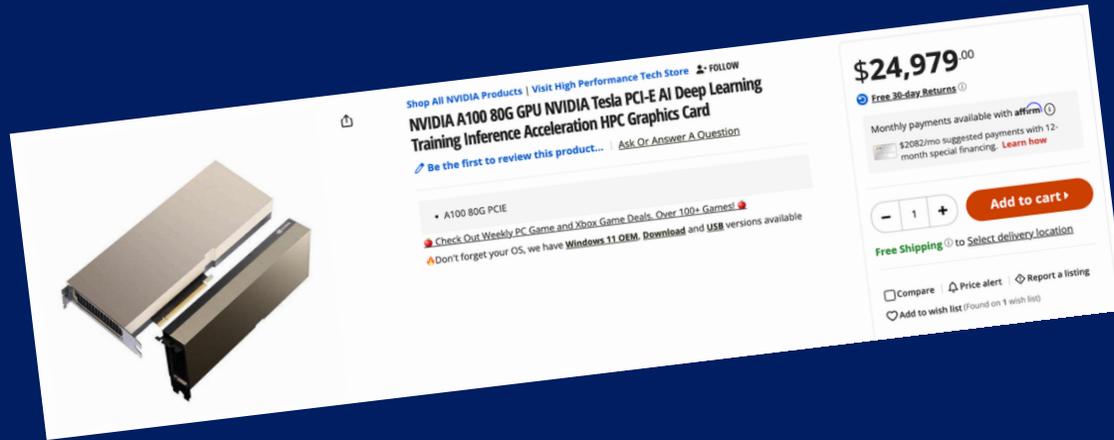
🚫 You require 5 GPUs for Full Training Adam

⚠ You require 2 GPUs for LoRa Fine-tuning (2.0%)

	float32	float16/bfloat16	int8
Total Size (GB)	27.96	13.98	6.99
Inference (GB)	33.55	16.77	8.33
Training using Adam (GB)	111.83	55.92	27.96
LoRA Fine-Tuning (GB)	34.99	18.22	9.83

Number of GPUs required for GeForce RTX 4090 (24.0 GB, 2022)





Fine-tuning models

Create your own custom models by fine-tuning our base models with your training data. Once you fine-tune a model, you'll be billed only for the tokens you use in requests to that model.

[Learn about fine-tuning >](#)

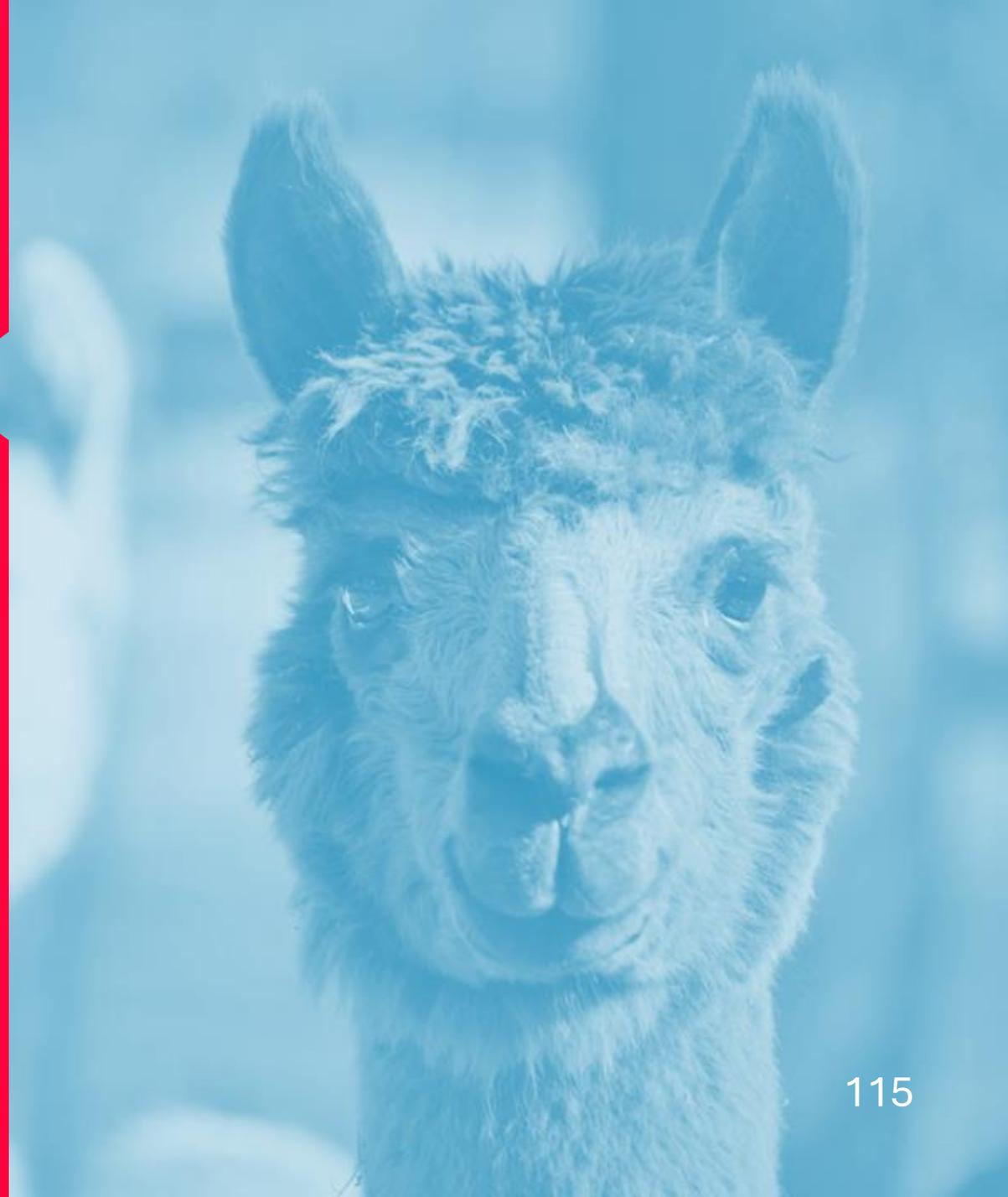
Model	Pricing	Pricing with Batch API*
gpt-4o-2024-08-06**	\$3,750 / 1M input tokens \$15,000 / 1M output tokens	\$1,875 / 1M input tokens \$7,500 / 1M output tokens
gpt-4o-mini-2024-07-18**	\$25,000 / 1M training tokens \$0.300 / 1M input tokens \$1.200 / 1M output tokens	\$0.150 / 1M input tokens \$0.600 / 1M output tokens
gpt-3.5-turbo	\$3,000 / 1M training tokens \$3,000 / 1M input tokens \$6,000 / 1M output tokens	\$1,500 / 1M input tokens \$3,000 / 1M output tokens
davinci-002	\$8,000 / 1M training tokens \$12,000 / 1M input tokens \$12,000 / 1M output tokens	\$6,000 / 1M input tokens \$6,000 / 1M output tokens
babbage-002	\$6,000 / 1M training tokens \$1,600 / 1M input tokens \$1,600 / 1M output tokens	\$0,800 / 1M input tokens \$0,800 / 1M output tokens

As a result, many researchers and NLP hobbyists find themselves facing a dilemma....

- Should they purchase the necessary hardware to run training and inference cycles for their desired LLM architecture?
- Or, should they purchase API credits to access an externally hosted LLM?

Open and Closed Models

- **Open Models:** LLMs that are open-sourced, meaning that you can download their source code and make updates, retrain them, and run inference using them (provided that you have the necessary hardware)
 - Llama
 - BLOOM
 - BERT
- **Closed Models:** LLMs that are *not* open-sourced, meaning that you can only access them through API calls.
 - Recent GPT models
 - Gemini
 - Claude



Which is better to use: open or closed models?

Open Models

- Advantages:
 - Enhanced data security
 - Only downstream cost is power consumption
 - Customizable
 - Transparent
- Disadvantages:
 - Big upfront cost (hardware!)
 - Requires systems expertise
 - Ample opportunity for bugs to emerge

Closed Models

- Advantages:
 - No local hardware requirements
 - LLM host takes care of messy backend issues
 - Easy to switch between LLM architectures
 - Straightforward to use
- Disadvantages:
 - Costs are incurred with each use
 - Data is sent to third-party
 - Black box

Cases that may suggest using one or the other include....

Scenario: You are working in a well-established research lab with a GPU cluster, and you want to experiment with many variations of an LLM

- Recommendation: **Open models** will allow you to leverage your existing resources and more fully customize your model

Scenario: You are just starting out exploring LLMs, and you have an idea that you want to test for a personal project

- Recommendation: **Closed models** will abstract away a lot of the complicated setup required to train and use LLMs, and the number of API calls you'll need for a personal project likely won't incur massive expenses

Scenario: You're a doctor and you're interested in whether LLMs can effectively find important details in your patients' medical records

- Recommendation: **Open models** will allow you to keep your data in your local environment---you don't want to risk medical records being used for later model training by a third-party entity!

Scenario: You're working in a well-established research lab and you'd like to use an LLM to generate synthetic data for downstream classifier training

- Recommendation: **Either** type of model will work! Open models are more replicable if your lab is equipped to run them, whereas closed models may produce a bit higher-quality output and are unlikely to be expensive for generating a fixed number of data samples

How can we release models and data to the greater community?

- To release **data** to the community, there are a few ways to ensure that others can find and access it:
 - Host the data on a well-known data repository
 - HuggingFace Hub: <https://huggingface.co/datasets>
 - LRE Map: <https://lremap.elra.info/>
 - Publish a paper about the data
 - International Conference on Language Resources and Evaluation: <https://recoling-2024.org/>
 - arXiv (for non peer-reviewed work): <https://arxiv.org/>
 - Create a datasheet with clear guidelines and licensing terms

How can we release models and data to the greater community?

- To release **models** to the community, you can:
 - Host the code in a publicly accessible repository with clear documentation
 - GitHub
 - HuggingFace Hub: <https://huggingface.co/docs/hub/en/index>
 - Zenodo: <https://zenodo.org/> (gives you citeable DOI, which is helpful for many researchers)
 - Establish clear protocols for use, reuse, modification, and attribution
 - Give people a bibliography entry that they can easily add as a reference!
 - Fill out a responsible AI checklist with full information available about library versions, runtime environment, hardware requirements, and hyperparameters so that others can easily start using your model in their own pipelines

Summary: Masked Language Modeling and Practical Guidelines for Data-Driven NLP

- Bidirectional Transformers are used to train LLMs with **masked language modeling** objectives
- Whereas causal LLMs are often viewed as **decoder-only** models, masked language modeling LLMs are often viewed as **encoder-only** models
- **BERT** is a popular bidirectional Transformer architecture
- When we share datasets publicly, it is a best practice to also share a **datasheet**
- When we share models publicly, it is a best practice to share code, hyperparameters, guidelines for use, and other information that others may need to **replicate** the work
- LLMs may be available as **open** or **closed** models, and these models may be best-suited to different types of problems or computing environments