Generative AI

Natalie Parde, Ph.D.
Department of Computer Science
University of Illinois Chicago

"generative ai"

News | Images | Videos | Perspectives | Examples | Startups | Use cases | Free | Algorithms

About 482,000,000 results (0.40 seconds)

Jan. 26

MUSIC

Explicit Deepfake Images of Taylor Swift Elude Safeguards and Swamp Social Media

Fans of the star and lawmakers condemned the images, probably generated by artificial intelligence, after they were shared with millions of social media users.

By Kate Conger and John Yoon

June 14, 2023

TECHNOLOGY

Generative A.I. Can Add $4.4 Trillion in Value to Global Economy, Study Says

The report from McKinsey comes as a debate rages over the potential economic effects of A.I.-powered chatbots on labor and the economy.

By Yiwen Lu

PRINT EDITION A.I. Can Add $4.4 Trillion Worldwide, Report Says | June 14, 2023, Page B6

Jan. 25

TECHNOLOGY

The Sleepy Copyright Office in the Middle of a High-Stakes Clash Over A.I.

The office is reviewing how centuries-old laws should apply to artificial intelligence technology, with both content creators and tech giants arguing their cases.

By Cecilia Kang

Jan. 25

THE LEARNING NETWORK

What Students Are Saying About Learning to Write in the Age of A.I.

Does being able to write still matter when chatbots can do it for us? Teenagers weigh in on an essay from Opinion.

By The Learning Network

Feb. 1

BUSINESS

Generative A.I.'s Biggest Impact Will Be in Banking and Tech, Report Says

For some companies, the new technology is an opportunity to enhance productivity and profit. Will their workers benefit as well?

By Steve Lohr

PRINT EDITION
February 2, 2024

Sept. 28, 2023

BUSINESS

IBM Tries to Ease Customers' Qualms About Using Generative A.I.

The company will assume the legal risk of businesses that use its A.I. systems and will publish the technology's underlying data.

By Steve Lohr

PRINT EDITION IBM Tries Assuring Customers Generative A.I. Is a Safe Bet | September 29, 2023, Page B7

Feb. 1

TECHNOLOGY

Can This A.I.-Powered Search Engine Replace Google? It Has for Me.

A start-up called Perplexity shows what's possible for a search engine built from scratch with artificial intelligence.

By Kevin Roose

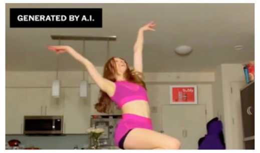
Feb. 1

MEDIA

A.I. Fuels a New Era of Product Placement

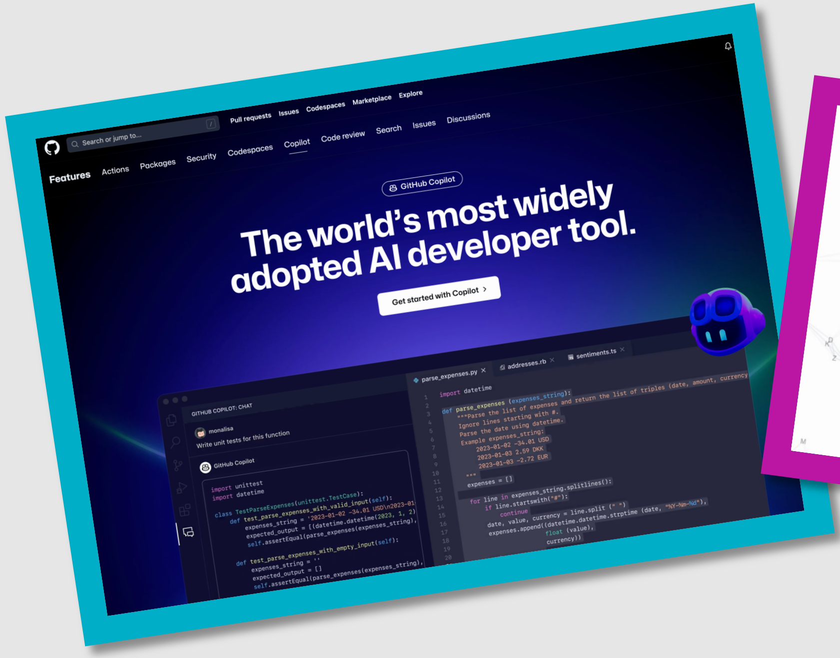
Realistic-looking shampoo bottles and seltzer cans are popping up on videos from digital creators on TikTok and YouTube in a new form of old advertising.

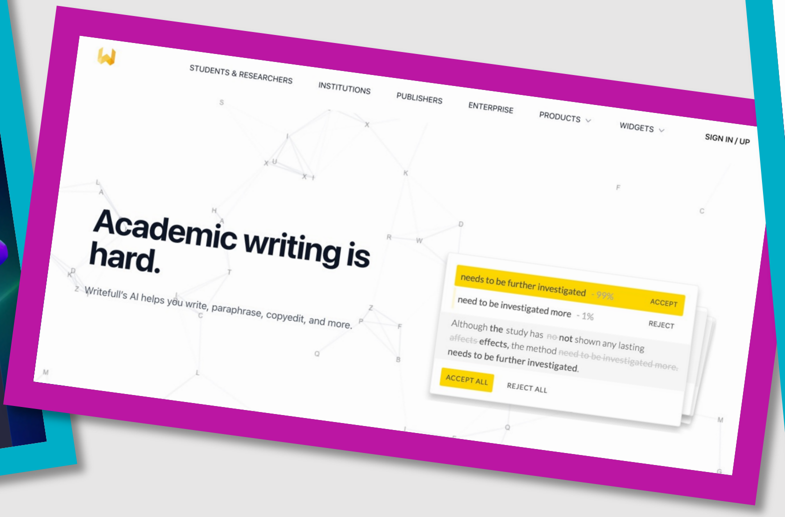
By Sapna Maheshwari

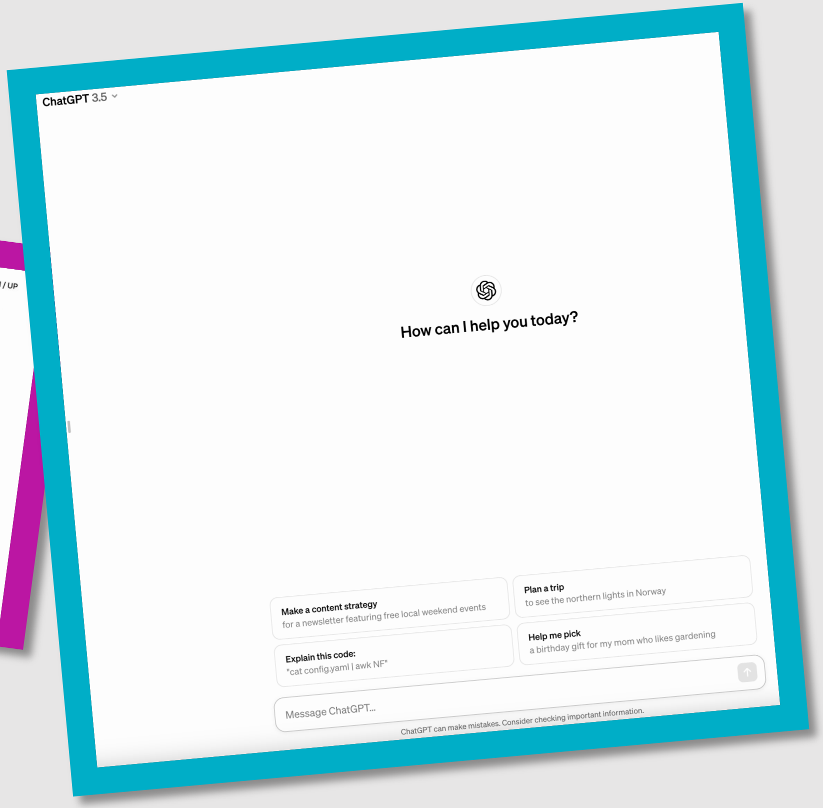


What is generative AI?

- Broadly speaking: Any machine learning model that focuses on generating output rather than categorizing or scoring input data
- More narrowly: Generally involves a very large, pretrained language model (typically referred to as a large language model, or LLM)
  - Can also involve multimodal models, to interpret or generate non-text data







Generative AI is becoming pervasive across applications!

Large Language Models

- What is “large”?
  - Not clearly defined, but generally speaking, anything “BERT-sized” (~110 million parameters) or larger
- Trained on massive quantities of text data to predict which word(s) should appear, given a context
- Can theoretically use any architecture that works for this setting, but in practice, modern LLMs are Transformer models

How are LLMs pretrained?

- Can be pretrained with numerous objectives
  - Masked language modeling
  - Next sentence prediction
  - Autoregressive generation
- Different pretraining objectives are useful for different purposes
  - Pretraining for masked language modeling may produce LLMs that are especially well-suited for classification
  - Pretraining for autoregressive generation may produce LLMs that are especially well-suited for longer-form generation tasks

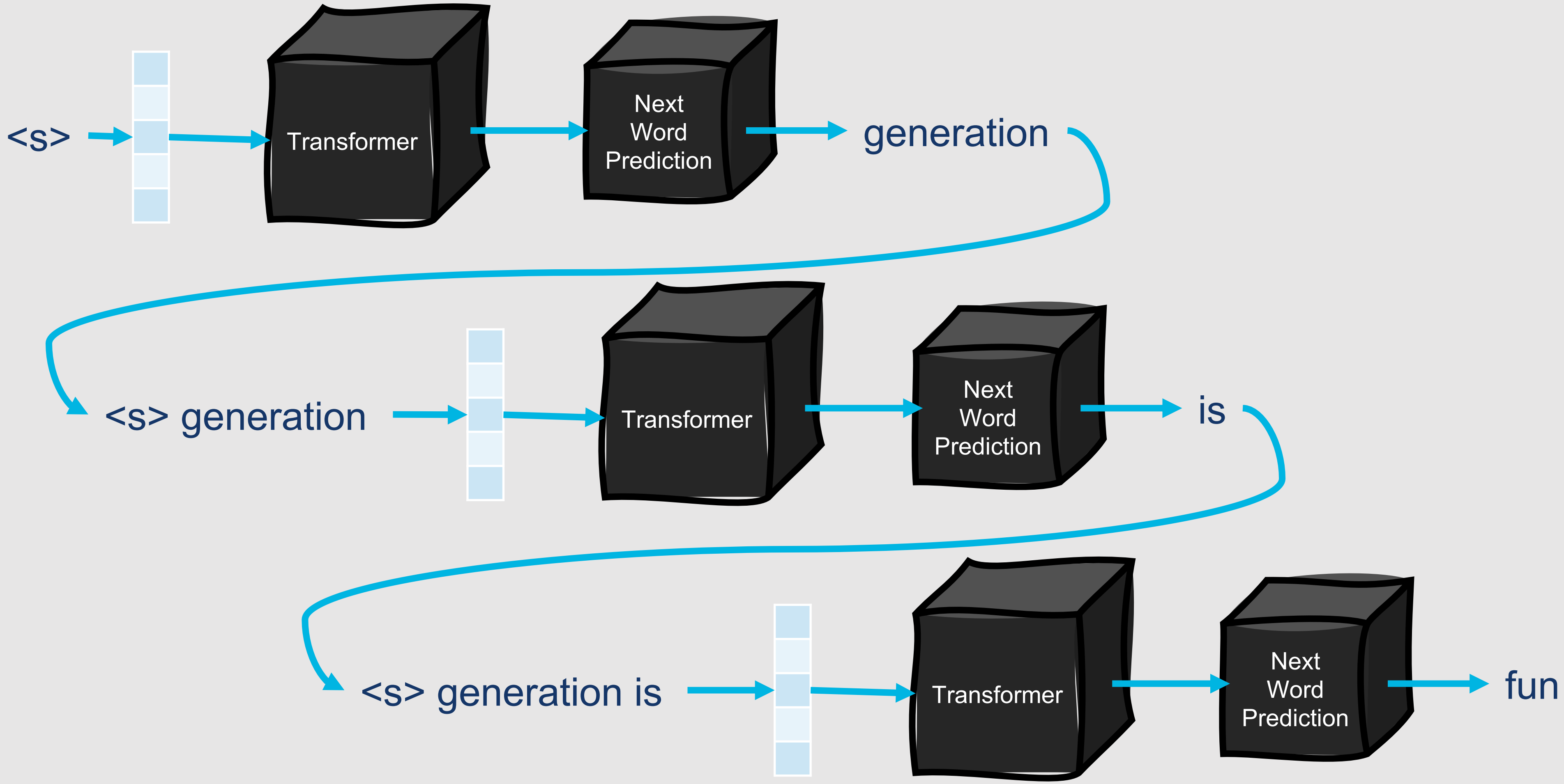
What’s most popular right now?

- The most popular LLMs right now (e.g., GPT-X or LLaMa) are pretrained for autoregressive generation
  - Given the sequence of words that have been generated so far, decide which word should come next

Generative Pretrained Transformer

G P T

Autoregressive Generation



Is this a step back?

- First came autoregressive generation, then came masked language modeling, then came …autoregressive generation again?
  - Autoregressive generation *without* instruction tuning is only useful for limited purposes (e.g., autocomplete)
  - Autoregressive generation + **instruction tuning** + **reinforcement learning with human feedback** (+ better **prefixes**) is a very recent development, and much more useful!

In fact, these recent developments have ushered in a new training paradigm.

- Why?
  - Fine-tuning pretrained models to perform new tasks works very well in many cases, but it still **requires that you have a reasonably large supervised training set** for the target task
  - In some cases, we only have a very tiny amount of training data (or none at all) for our target task!

Rule-Based Era
- Prior to ~1990s

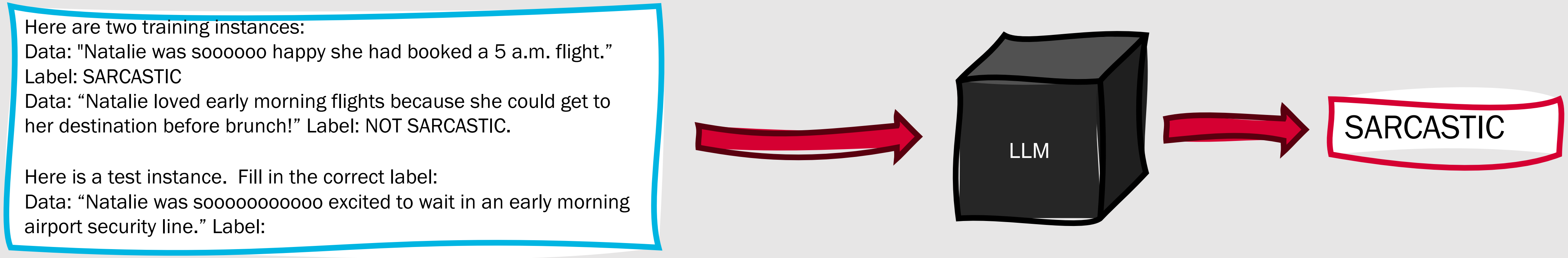
Statistical and (Early) Neural Era
- 1990s to 2010s

Pretrain and Finetune Era
- Late 2010s to present

Pretrain and Prompt Era
- Early 2020s to present

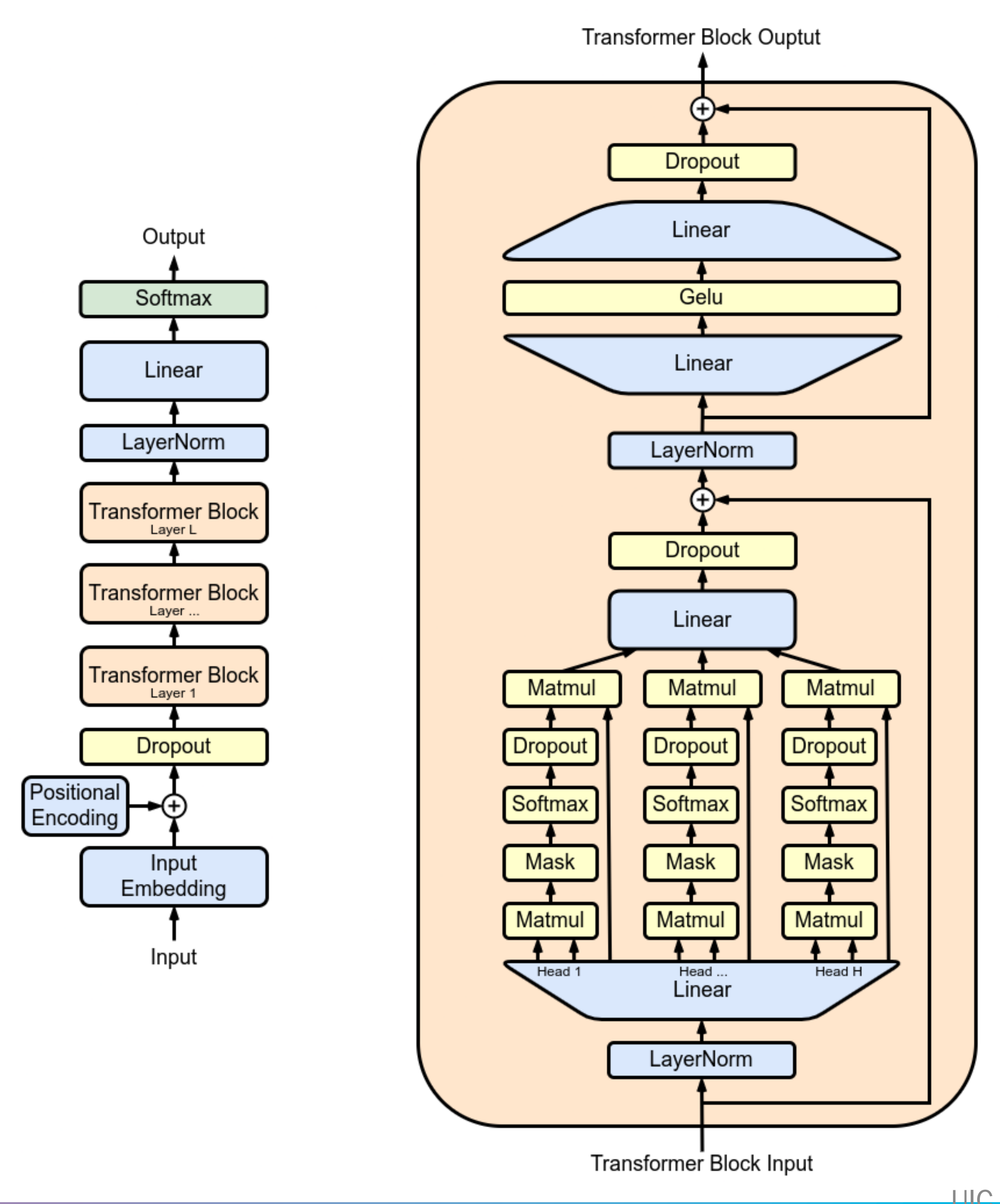
Introducing: Pretrain (and Optionally Fine-Tune) and Prompt

- Intuition:
  - If we take LLMs that have been pretrained on a wide variety of language data, we can **prompt** them to produce the correct labels or output for new tasks



This new paradigm has seen remarkably rapid uptake in the NLP community!

	# Full, Main Conference Papers with "Prompt" in Title
ACL 2022	22
EMNLP 2022	41
ACL 2023	36
EMNLP 2023	44



At the core of most recent work are **generative pretrained Transformers** (**GPTs**).

- Original GPT architecture was published in 2018: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf
  - Transformer *decoder* model
  - 12 Transformer blocks
  - 12 attention heads per self-attention layer
  - Trained on BooksCorpus
    - 7000 books

Popular Large (Generative) Language Models

- Since the original GPT, these models have grown increasingly larger!
  - GPT-X
    - ~0.5 trillion tokens of pretraining data (last reported for GPT-3)
  - LLaMa
    - ~1.4 trillion tokens of pretraining data
- How much data *is* a trillion tokens?
  - ~15,000,000 books!

Open vs. Closed Models

GPT-4

With broad general knowledge and domain expertise, GPT-4 can follow complex instructions in natural language and solve difficult problems with accuracy.

Learn about GPT-4

Model	Input	Output
gpt-4	$0.03 / 1K tokens	$0.06 / 1K tokens
gpt-4-32k	$0.06 / 1K tokens	$0.12 / 1K tokens

GPT-3.5 Turbo

GPT-3.5 Turbo models are capable and cost-effective.

`gpt-3.5-turbo-0125` is the flagship model of this family, supports a 16K context window and is optimized for dialog.

`gpt-3.5-turbo-instruct` is an Instruct model and only supports a 4K context window.

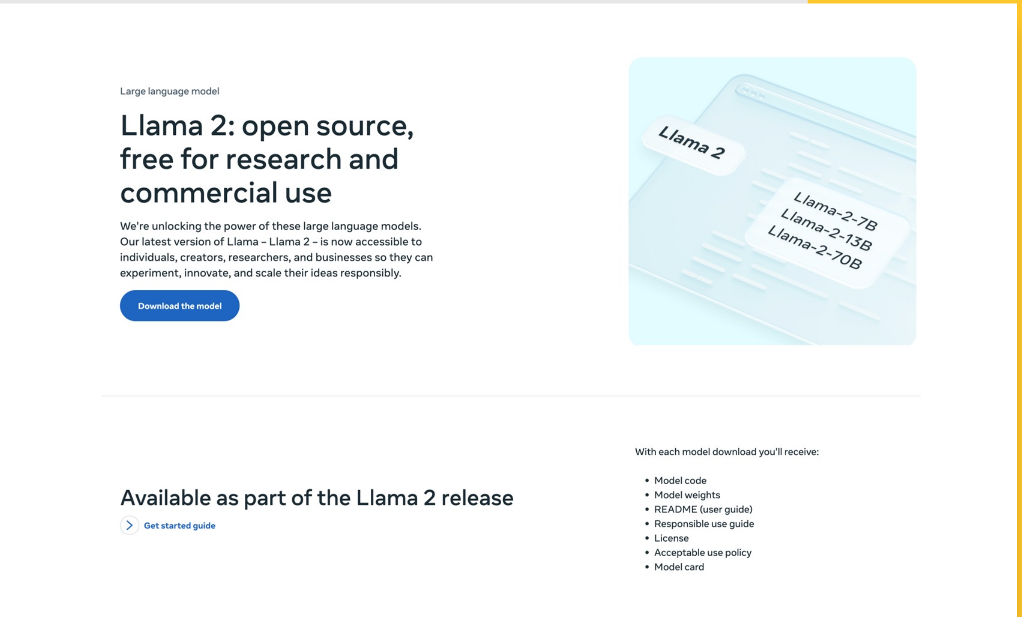
Learn about GPT-3.5 Turbo ↗

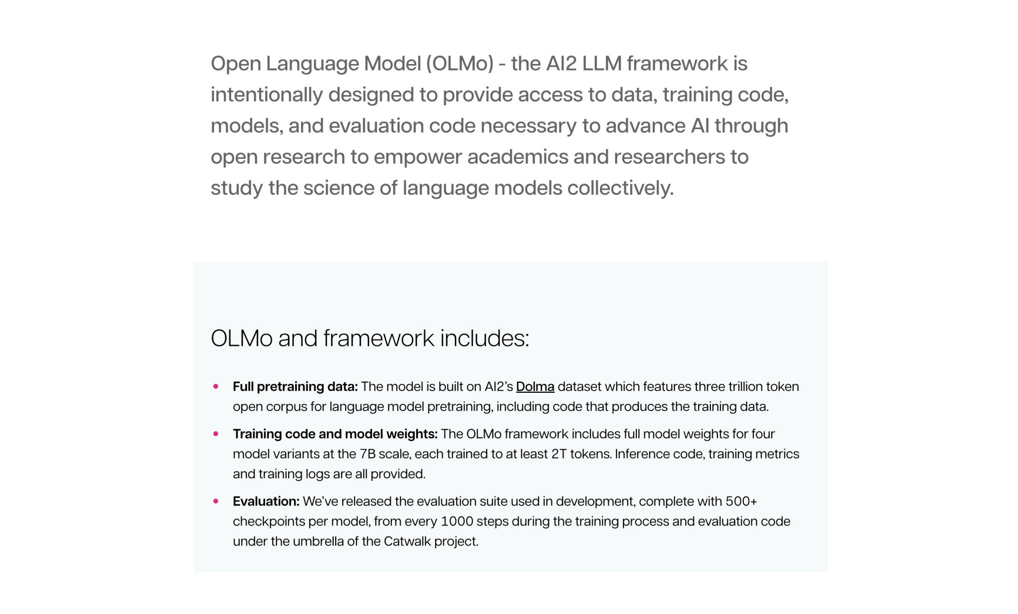
Model	Input	Output
gpt-3.5-turbo-0125	$0.0005 / 1K tokens	$0.0015 / 1K tokens
gpt-3.5-turbo-instruct	$0.0015 / 1K tokens	$0.0020 / 1K tokens

- Many popular high-performing LLMs are closed models
  - Full model cannot be modified or directly accessed by researchers
  - Details about training data and architecture may be scarce
  - Accessible via paid API
  - Example: GPT-4

Open vs. Closed Models

- However, very recent interest (and helpful efforts from community members!) have led to the public release of several open-source LLMs
  - Fully accessible and modifiable
  - Architecture is fully explorable
  - Free!
  - Examples:
    - Llama 2: https://llama.meta.com/llama2
    - OLMo: https://allenai.org/olmo





LLM Resources

- Open LLM Leaderboard: https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard
- A Survey of Large Language Models
  - Paper: https://arxiv.org/abs/2303.18223
  - Repository: https://github.com/RUCAIBox/LLMSurvey
- Generative models on the Hugging Face model hub: https://huggingface.co/models?pipeline_tag=text-generation&sort=trending

Instruction Tuning

- Contemporary LLMs work well at solving a diverse range of tasks, despite their often being pretrained on tasks far from the end application goal
  - Example: general-domain autoregressive language modeling
- We ideally want LLMs to *optimally* follow our directions to solve desired task(s)
- How can we improve the LLM's ability to do this?
  - Instruction Tuning: An emerging technique to align pretrained generative language models with end application goals

Why perform instruction tuning?

Advantages of instruction tuning:

- Bridges the gap between standard language modeling pretraining tasks and end user task goals
- Encourages more controllable and predictable model behavior
- Promotes computational efficiency
- Facilitates domain adaptation

However, instruction tuning is challenging….

- Few high-quality instruction tuning datasets are available
  - Instructions should be diverse, creative, and cover the desired target outcomes
- No guarantee that the tuned model will generalize beyond the tasks covered in the instruction tuning dataset
- May only learn surface-level patterns associated with task data (rather than real characteristics of the task)

How does instruction tuning work?

- Many variations!
- In general:
  - LLMs are fine-tuned using (INSTRUCTION, OUTPUT) pairs
    - **Instruction:** Human input to the model
    - **Output:** Desired task output
  - Fine-tuning is supervised
    - Given the input, learn to predict the output autoregressively based on the instruction tuning data

Building Instruction Tuning Datasets

- Components:
  - Instruction (natural language text sequence)
    - General and specific task details
  - (Optional) supplemental context (natural language text sequence)
    - Related background information and/or demonstrations
  - Output (natural language text sequence)
    - Generated output label or text sequence

What does this look like?

Come up with some good acronyms for a new instruction tuning method that uses less manual data by augmenting the training set with synthetic generated data

General instruction (consistent across all training inputs)

Specific input for this data instance

What does this look like?

Come up with some good acronyms for a new instruction tuning method that uses less manual data by augmenting the training set with synthetic generated data
Acronyms should use the first letters of important words for the method. For example:
Method: an approach to interpret metaphors by mapping them to conceptual metaphor clusters
Acronym: MIRAGE: Metaphor Interpretation by Recognizing Associated Groupings and Expressions

Supplemental context

What does this look like?

Come up with some good acronyms for a new instruction tuning method that uses less manual data by augmenting the training set with synthetic generated data
Acronyms should use the first letters of important words for the method. For example:
Method: an approach to interpret metaphors by mapping them to conceptual metaphor clusters
Acronym: MIRAGE: Metaphor Interpretation by Recognizing Associated Groupings and Expressions

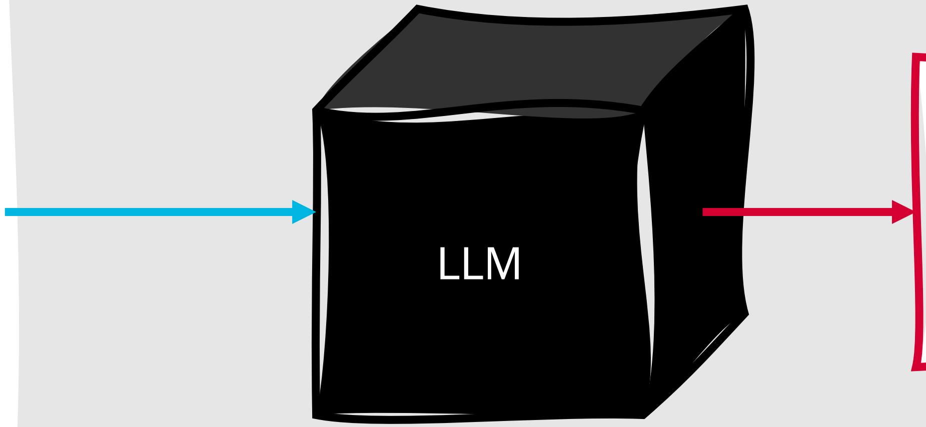
What does this look like?

Come up with some good acronyms for a new instruction tuning method that uses less manual data by augmenting the training set with synthetic generated data

Acronyms should use the first letters of important words for the method. For example:

Method: an approach to interpret metaphors by mapping them to conceptual metaphor clusters

Acronym: MIRAGE: Metaphor Interpretation by Recognizing Associated Groupings and Expressions



SYNTHETIC: Synthetic Yield for Novel Instruction Tuning and Human-less Training

Methods for Dataset Construction

- Data Integration
  - Develop templates to transform (text, label) pairs to (INSTRUCTION, OUTPUT) pairs
  - Collect relevant data from existing natural language datasets and apply the templates to that data
- Data Generation
  - Collect instructions
    - May be manually constructed or automatically generated by paraphrasing seed instructions
  - Generate outputs
    - Pass instructions to an LLM and collect the outputs for use in the dataset

Instruction tuning datasets created using data integration approaches:

Natural Instructions	100K+ (INSTRUCTION, INSTANCE) pairs spanning many diverse NLP tasks "Instance" elements are themselves (input, output) pairs Instructions refer to the specific task, whereas (input, output) pairs are sourced directly from existing NLP datasets https://github.com/allenai/natural-instructions
Public Pool of Prompts (P3)	(INPUT, ANSWER CHOICE, TARGET) triples integrating data across 170 English-language NLP datasets Input ⟶ Instruction https://huggingface.co/datasets/bigscience/P3
xP3	Cross-lingual extension of P3 16 tasks spanning 46 languages (still using English prompts) https://huggingface.co/datasets/bigscience/xP3
Flan 2021	(INPUT, TARGET) pairs created from 62 popular NLP benchmarks Instruction and target templates were manually constructed and instances from the benchmark datasets were then used to fill the templates https://github.com/google-research/FLAN
LIMA	1000 carefully curated (INSTRUCTION, RESPONSE) pairs Designed to demonstrate that only a small number of high-quality instruction tuning examples are needed to produce high performance https://huggingface.co/datasets/GAIR/lima

Instruction tuning datasets created using data generation approaches:

Unnatural Instructions	(INSTRUCTION, INPUT, CONSTRAINTS, OUTPUT) tuples seeded from a third-party instruction tuning dataset and automatically expanded using InstructGPT
	https://github.com/orhonovich/unnatural-instructions
Self-Instruction	(INSTRUCTION, INPUT, OUTPUT) triples seeded from 175 tasks and automatically expanded using InstructGPT
	https://github.com/yizhongw/self-instruct
Baize	Multi-turn chat corpus where each turn includes a (PROMPT, RESPONSE) pair
	ChatGPT was used to generate data for both conversation parties
	https://github.com/project-baize/baize-chatbot

We have our dataset ...now, how does instruction tuning work?

- Commonly, instruction tuning involves a multi-stage process:
  1. Pretrain a model on a large, general-domain corpus
  2. Perform supervised fine-tuning of the model using an instruction tuning dataset
  3. Further tune the model using **reinforcement learning from human feedback (RLHF)**
- Some approaches use only supervised fine-tuning (SFT) or only RLHF

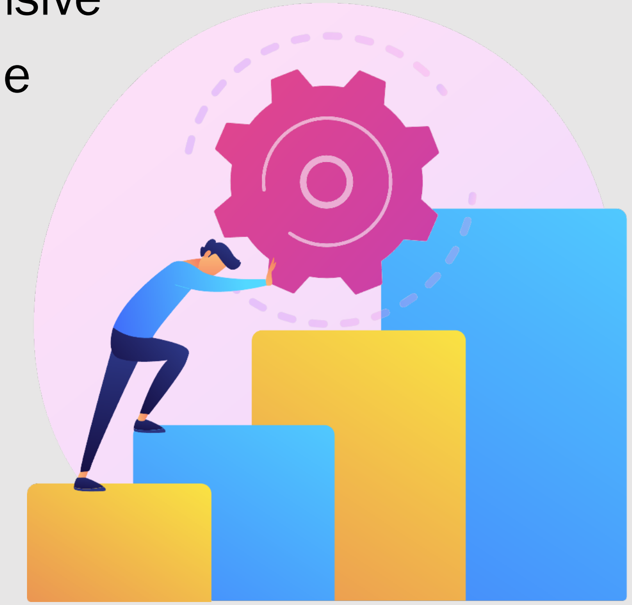
What are the advantages to including these tuning steps?

Pretraining a large language model ⟶ extremely resource-intensive

- Model learns an impressive amount of knowledge, but doesn't fully understand how to apply it

SFT or RLHF ⟶ less resource-intensive

- Big performance gains for little effort



Recall our pretraining objective!

- During pretraining, most popular contemporary LLMs are optimized for autoregressive generation
  - "Where is UIC"
    - …"in CS Rankings?"
    - …"on the map?"
    - …"recruiting its students from?"
    - …"UIC is located in Chicago, bordered by West Loop to the north and Pilsen to the south"
- This only tells us which words are most likely to follow a prefix (no specific incentive for the model to perform a task!)

Supervised Fine-Tuning

- Goal: Fine-tune the language model so that it completes the prefix by performing the task specified within the prefix
- How do we do this?
  - Instruction tuning data!
  - Show the model how to complete the prefix by performing the task, and it will start completing prefixes that way

Note: It's possible to skip pretraining and perform supervised instruction tuning from scratch; however, in practice this tends to produce lower performance than pretraining followed by supervised fine-tuning.

Supervised Fine-Tuning

- Fine-tuning process is similar to that observed in other transfer learning settings
  - Start with a pretrained model
  - Focus on updating the weights in the final layer(s)
  - Optimize weights using a cross-entropy loss function
    - Only the tokens in the completion of the prefix are considered when calculating loss

How much data is needed for supervised instruction fine-tuning?

- ~1000-100,000 samples
  - Most often, more than 10,000 samples
  - LIMA uses only 1000 carefully curated samples and prompts
    - https://openreview.net/pdf?id=KBMOKmX2he

Supervised fine-tuning is limited in several ways.

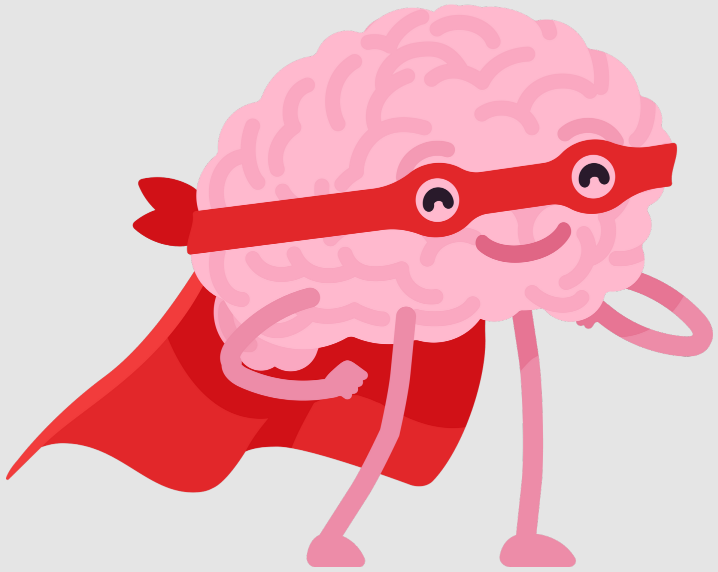
- There are many ways to complete a prefix by performing the task specified in the prefix, but some of those ways are much better than others
- Instruction tuning datasets provide demonstrations regarding how to perform the task specified in the prefix, but no ratings for those demonstrations!

We need....

- A scoring function that rates the quality of an (INSTRUCTION, OUTPUT) pair
- A way to use this scoring function to train LLMs to generate higher-performing outputs

RLHF to the rescue!

- Reinforcement learning from human feedback is a two-step process:
  1. Train a reward model to score (INSTRUCTION, OUTPUT) pairs
  2. Optimize the LLM to generate higher-scoring outputs for instructions



How does the reward model work?

- Goal: Output a score for an (INSTRUCTION, OUTPUT) pair
- In many ways, similar to other classification or regression problems
- However, challenges may include determining:
  - Where to obtain trustworthy scoring data
  - How to ensure that annotators agree on scores (can be highly subjective!)

Data Labeling for Reward Models

- Generally framed as a comparison task
  - Given two possible completions for a prefix, which is better?
- Thus, comparison data is structured as:
  - (INSTRUCTION, BETTER OUTPUT, WORSE OUTPUT)

Note: Comparison data tends to have reasonably good agreement, but comparing outputs for an instruction is still often very subjective.

How do we train a reward model to predict scores, given comparison data?

- Train the model to maximize a score difference between the better and worse outputs
- Reward models can be trained for scratch, or initialized using the supervised fine-tuned model
    - Initializing using the SFT model tends to work better (intuition: the reward model should be at least as powerful as the underlying LLM)

How much comparison data is needed?

- ~100,000-1,000,000 comparison samples
  - Should include numerous comparisons for each prompt

- As an example, InstructGPT used:
  - 50,000 prompts
  - Each with 4-9 outputs
  - Organized into (INSTRUCTION, BETTER OUTPUT, WORSE OUTPUT) pairs, this meant that there were between 6-36 comparisons per prompt
  - InstructGPT paper: https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf

We've trained our reward model …now, how do we optimize our LLM?

Objective: Improve the performance of the fine-tuned model such that it generates outputs that maximize the scores assigned by the reward model

Common approach for this: A reinforcement learning algorithm known as **proximal policy optimization (PPO)**

Proximal Policy Optimization

Prompts are randomly selected from a distribution → These prompts are passed into an LLM → Output from the LLM is scored using a reward model

Additional constraint added:

- Output should be relatively similar to the output generated by the SFT model (prior to reinforcement learning from human feedback) and to the output generated by the original pretrained model

Why include the additional constraint?

- The reward model has seen very few outputs for a given instruction
- For unknown (INSTRUCTION, OUTPUT) pairs, the reward model might mistakenly predict very high or very low scores
- This may consequently bias the model towards generating (mistakenly) highly scored outputs

In reinforcement learning terms....

- **Action Space:** The tokens in the LLM's vocabulary
  - Taking an action: Choosing a token to generate
- **Observation Space:** The distribution over all possible instructions
- **Policy:** The probability distribution over all actions that can be taken, given an observation
  - An LLM is a policy: It predicts which token should be generated next, given a prefix

- How much data is used for PPO?
  - Approximately 10,000 to 100,000 instructions

Limitations of RLHF

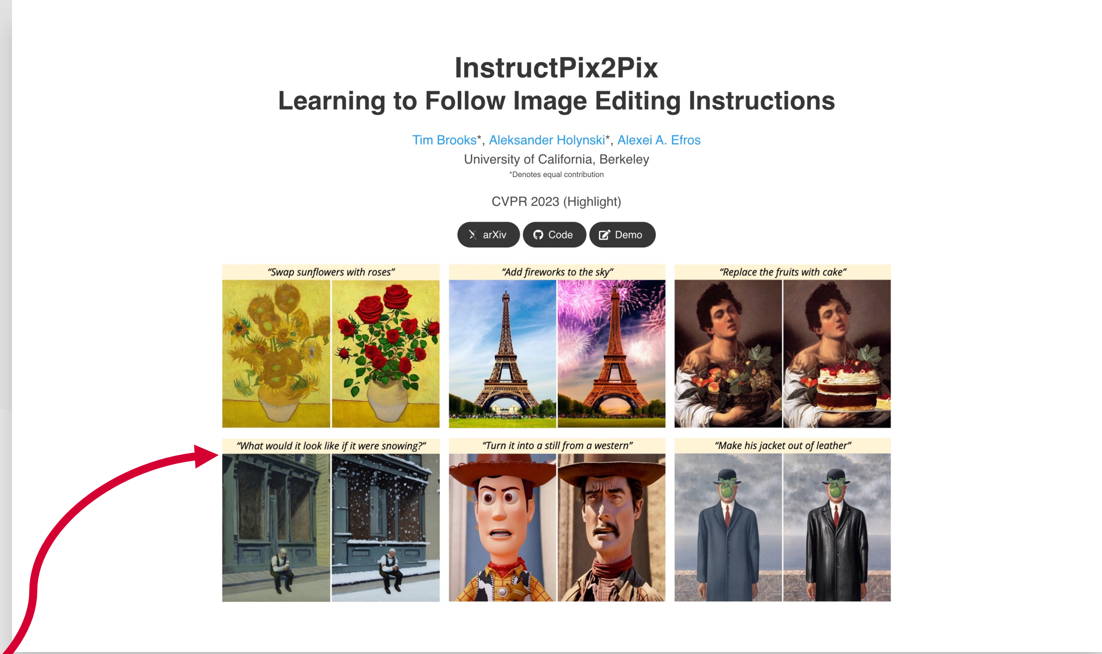
- RLHF has only very recently been applied to language modeling setups, and much is still unknown!
- Big issue currently: **Hallucination**
  - LLMs tend to make up information (and phrase misinformation confidently!)
- Why does RLHF promote hallucination?
  - Two leading hypotheses:
    - LLMs don't understand the causes and effects of their actions
    - LLMs' internal knowledge is misaligned with human knowledge

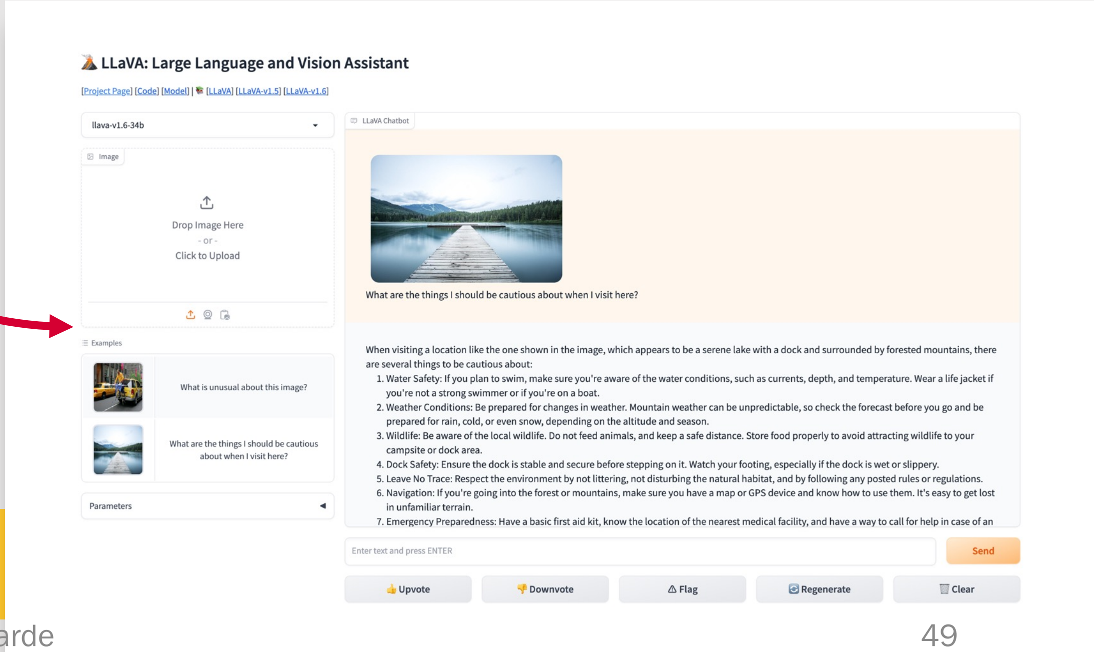
We've trained our model using supervised fine-tuning and RLHF …now what?

- Some instruction tuning settings require special considerations:
  - Multimodal applications
  - Specialized application domains
- In these cases, we need to adapt this process to accommodate extra information

Multimodal Instruction Tuning

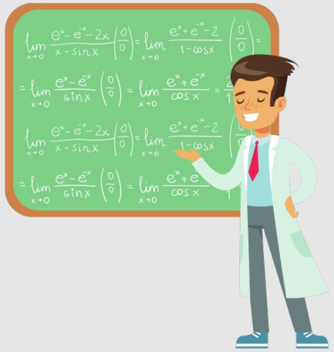
- Typically involves extra steps
  - **InstructPix2Pix:** First fine-tunes an LLM to predict (IMAGE CAPTION, EDIT INSTRUCTION, UPDATED IMAGE CAPTION) triples, and then uses a text-to-image model to convert (IMAGE CAPTION, UPDATED IMAGE CAPTION) pairs to image pairs
    - https://github.com/timothybrooks/instruct-pix2pix
  - **LLaVA:** Fine-tunes a vision encoder and an LLM decoder using a vision-language instruction tuning dataset
    - https://llava-vl.github.io/





Domain-Specific Instruction Tuning

- Use a specialized, domain-specific instruction tuning process:
  - Dialogue
    - InstructDial: https://github.com/prakharguptaz/Instructdial
  - Information Extraction
    - InstructUIE: https://github.com/BeyonderXX/InstructUIE
  - Writing
    - CoEdIT: https://github.com/vipulraheja/coedit
    - CoPoet: https://github.com/vishakhpk/creative-instructions
  - Medical
    - ChatDoctor: https://github.com/Kent0n-Li/ChatDoctor
  - Arithmetic
    - Goat: https://github.com/liutiedong/goat



Instruction tuning can be costly and time-consuming. How can we make this process more efficient?

- Carefully optimize as few parameters as possible!
- Ways to do this:
  - **Addition Methods:** Add trainable parameters
    - Adapter tuning: https://proceedings.mlr.press/v97/houlsby19a/houlsby19a.pdf
    - Prompt-based tuning: https://aclanthology.org/2021.eacl-main.20.pdf
  - **Specification Methods:** Specify which parameters to tune and which to freeze
    - BitFit: https://aclanthology.org/2022.acl-short.1.pdf
  - **Reparameterization Methods:** Prior to tuning, reparameterize model weights so that they're more efficient
    - LoRA: https://openreview.net/pdf?id=nZeVKeeFYf9

Low-Rank Adaptation (LoRA)

- Reduces the number of trainable parameters during instruction tuning by introducing update matrices to existing weights, and focusing only on training the update matrices
  - Capable of reducing trainable parameters by 10,000x, and memory usage by 3x!
  - Why does this work well?
    - Freezes previously trained weights → Ensures that the language model retains existing knowledge while adapting to new data
    - Fewer parameters → Better portability
    - Incorporated into attention layers of original model → Helps control the extent to which the model adjusts to new data

How does LoRA work?

- Update matrices are **rank decomposition matrices**
  - Recall from linear algebra ...basically, a rank decomposition matrix approximates a given matrix with a **lower-rank matrix**
  - Original paper recommends a minimum rank = 8
    - More complex datasets require higher ranks
    - Higher ranks require more compute resources
  - Full supervised fine-tuning would have a rank equal to the model's hidden layer size

Matrix Rank: How many rows/columns can't be made from other rows/columns? For example, the rank of $\begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$ is 1, since the second row is just double the first row (no new information!)

Additional Hyperparameters in LoRA

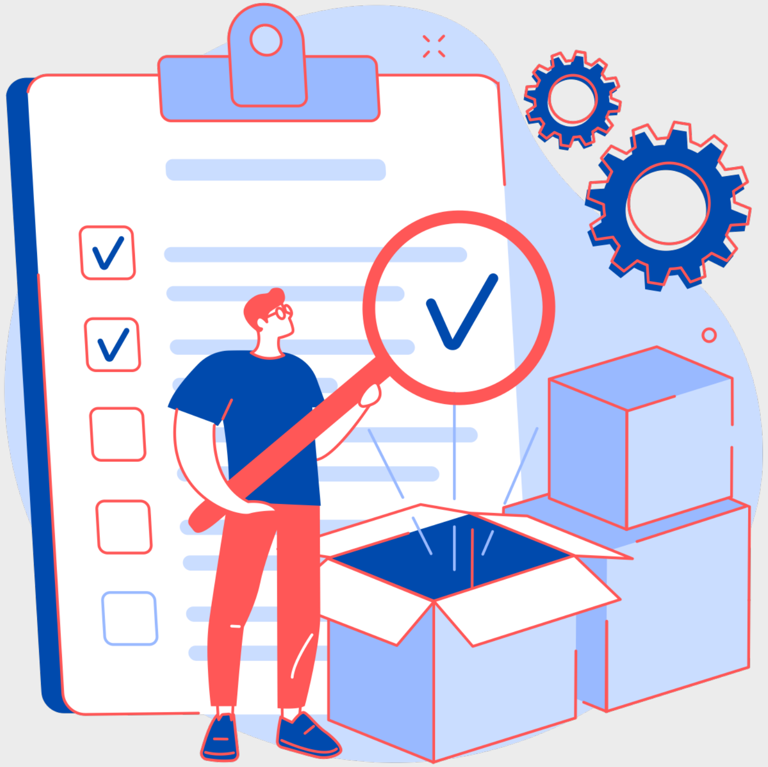
- **Alpha:** Scaling factor
  - Determines the extent to which the model is adapted to new data, by adjusting the contribution of the update matrices during training
  - Lower values → Maintain existing knowledge more than higher values
- **Target Modules:** Specific weights and matrices to be trained
  - May include:
    - Query vector projection matrix
    - Value vector projection matrix
    - Attention output projection matrix
    - Embedding token weights
    - Output layer
    - And many more!

Can we further improve efficiency beyond LoRA?

- **QLoRA: Quantized Low Rank Adapters**
  - Backpropagates gradients through a frozen quantized LLM into Low Rank Adapters
  - Improves efficiency over LoRA by:
    - Introducing a new data type, **4-bit NormalFloat**, that is information theoretically optimal for normally-distributed weights
    - Quantizing the quantization constants ("**Double Quantization**")
      - Quantization constant = Ratio of the maximum of the quantized range to the absolute maximum value of the original tensor
    - Managing memory spikes using **paged optimizers**
      - Paged optimizers = Hardware feature that allows you to move paged memory of optimizer states between the CPU and GPU
  - https://github.com/artidoro/qlora

What influences instruction tuning outcomes?

- Many factors!
  - Data quality
  - Data coverage
    - Greatly influences the extent to which the model can generalize



Challenges and Next Steps in Instruction Tuning

- Low-resource instruction tuning
- Evaluating instruction tuning data
- Understanding what is learned during instruction tuning

Low-Resource Instruction Tuning

- How much data is needed to perform high-quality instruction tuning (i.e., performance similar to the state-of-the-art)?
  - On average, performing instruction tuning with ~25% of the samples required for fully supervised models will outperform the state-of-the-art: https://arxiv.org/abs/2306.05539
    - In multitask learning settings, only ~6% of the data is needed!
  - LIMA performs well using only 1000 carefully selected samples
    - Positive implications for many low-resource languages and domains
    - https://openreview.net/forum?id=KBMOKmX2he

How can we ensure high-quality instruction tuning data?

- No standardized measure of instruction tuning data quality
- In general:
  - Human-written samples are higher quality but more expensive to obtain
  - AI-generated samples may be noisy or more biased
- When assessing quality:
  - Heuristics or pretrained models may result in noisy outputs
  - Human-annotated quality labels are preferred, but expensive
- Ensuring broad coverage of (INSTRUCTION, OUTPUT) pairs results in better performance!

What do instruction-tuned models *really* learn?

- Evidence suggests that much of what instruction-tuned models learn is attributable to superficial patterns (e.g., understanding output format and making informed guesses) rather than truly understanding the task
  - https://aclanthology.org/2023.acl-short.113.pdf
- There is currently no standardized way to evaluate instruction-tuned models!
  - Emerging metrics:
    - IFEval: https://github.com/google-research/google-research/tree/master/instruction_following_eval
    - LMEntry: https://github.com/aviaefrat/lmentry
    - M2C: https://github.com/google-research/multi-morph-checklist

Resources for Instruction Tuning

- Axolotl: https://github.com/OpenAccess-AI-Collective/axolotl
- LLaMA-Factory: https://github.com/hiyouga/LLaMA-Factory
- Open-Instruct: https://github.com/allenai/open-instruct

- Helpful Literature:
  - Instruction Tuning for Large Language Models: A Survey: https://arxiv.org/pdf/2308.10792.pdf
  - RLHF: Reinforcement Learning from Human Feedback: https://huyenchip.com/2023/05/02/rlhf.html

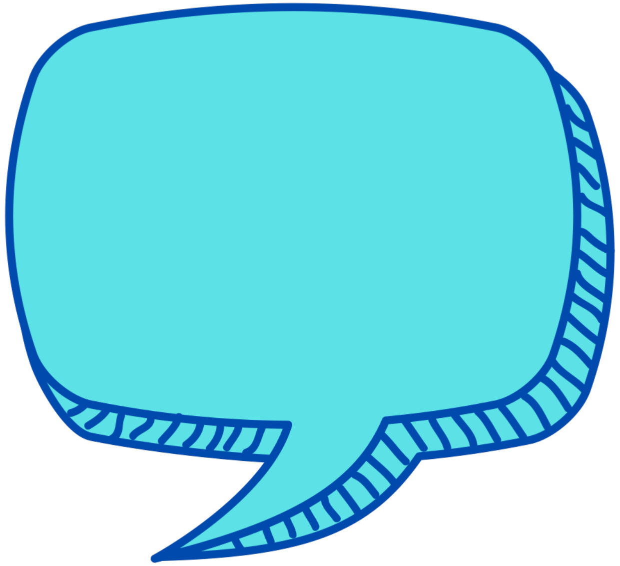
Summary: LLMs and Instruction Tuning

- Broadly speaking, text-based **generative AI** involves prompting **a large language model (LLM)** that has optionally been fine-tuned to follow a particular set of instructions
- **Instruction tuning** is done using (INSTRUCTION, OUTPUT) pairs
- Instruction tuning can involve both **supervised fine-tuning (SFT)** and **reinforcement learning from human feedback (RLHF)**
- In RLHF, a **reward model** learns to score (INSTRUCTION, OUTPUT) pairs and the LLM is then optimized to generate higher-scoring outputs for input instructions
    - One popular way to optimize the LLM is by using **proximal policy optimization (PPO)**
- **Low-Rank Adaptation (LoRA)** or its extension **QLoRA** can be used to greatly improve the efficiency of instruction tuning

Prompting

We've pretrained and instruction-tuned our LLM …how do we ensure the highest-quality results?

- Prefixes can greatly influence task performance!
- Prefixes used to elicit classification labels, scores, or other desired output from LLMs are typically referred to as **prompts**



How does prompting work?

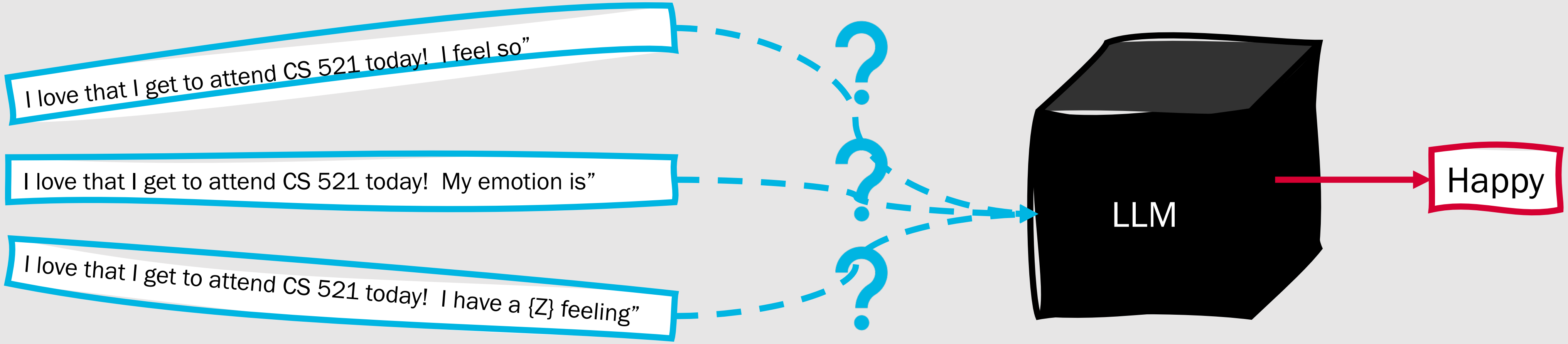
- Take a large model that has already been trained to model language
- Develop **prompt templates** for your task
  - Prompt templates can be manually or automatically constructed
- Develop an approach for **answer engineering**
  - Build an answer space (set of possible answers that your model may generate) 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

Why is this useful?

- Successful approaches using the pretrain and prompt paradigm are able to perform **few-shot** or even **zero-shot** learning for the target task
  - 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

In a way, prompting works in an opposite way from transfer learning setups….

- Transfer learning: Adjust the model's weights so that it works better with your task and data
- Prompting: Adjust (or, edit the description of) your task and data so that it works better with the model's weights



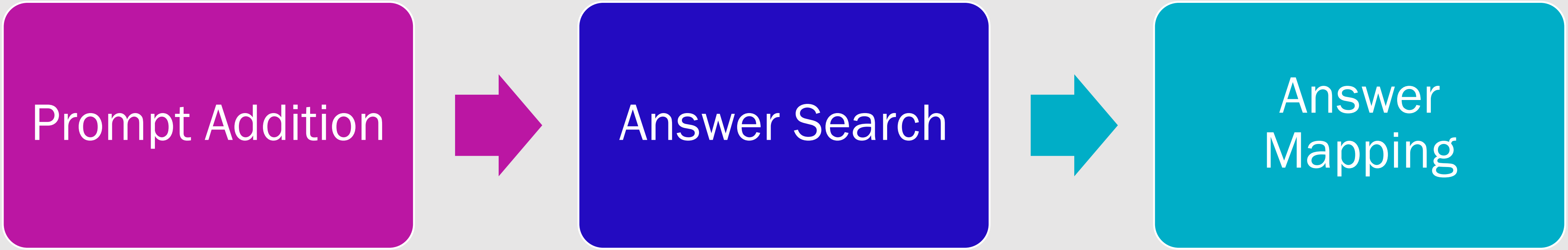
Formal Definition of Prompting

- In traditional supervised learning, our end goal is to model the probability that a given input $\mathbf{x}$ has a label $\mathbf{y}$ by learning a model with the optimal parameters θ
  - $P(\mathbf{y}|\mathbf{x};\theta)$
  - However, in cases where we have limited data, it is difficult to perform this learning process!
- In prompting, we instead model the probability of $\mathbf{x}$ existing in a language modeled using the parameters θ, and that in turn informs how we predict $\mathbf{y}$
  - $P(\mathbf{x};\theta)$

Prompt Engineering

- Systematically and empirically investigating the best prompts to produce the desired outcome is referred to as **prompt engineering**
- Can sometimes be counterintuitive!
  - Sometimes the "best" prompts look weird to human developers, but work well at encouraging the model to perform the task well

How do we predict the highest-performing label using prompting?



Prompt Addition

- Goal: Modify $\mathbf{x}$ into a prompt $\mathbf{x}'$
  - $\mathbf{x}' = f_{\text{prompt}}(\mathbf{x})$
- How does f_{prompt} typically work?
  - Apply a text template that has slots for the input x and an intermediate answer z
  - Fill the slot x in the text template
  - For example:
    - x = "I love watching CS 521 lectures"
    - Template "[x]. I feel [z]"
    - $\mathbf{x}' = f_{\text{prompt}}(\mathbf{x})$ = "I love watching CS 521 lectures. I feel [z]"

What about the intermediate answer slot?

- Intermediate answer slots can appear within the prompt or at the end of it, depending on the prompting technique
- Prompts that have slots to fill *within* the prompt are **cloze prompts**
  - “text [x] text [z] text”
- Prompts that have slots to fill only at the *end* of the prompt are **prefix prompts**
  - “text x text [z]”
- The number of [x] and [z] slots in the prompt can be flexible
- Template words are often natural language tokens, but they don’t have to be

Example Inputs, Templates, and Answers

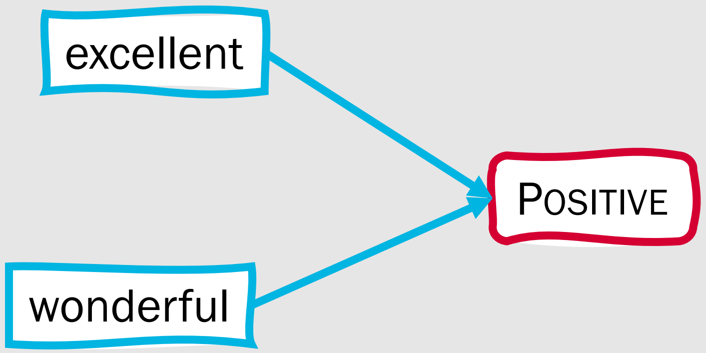
TASK	TEMPLATE	INPUT [X]	ANSWER [Z]
Topic Identification	[X] The text is about [Z]	She carefully read the journal article.	sports science …
Aspect-Based Sentiment Analysis	[X] How is the service? [Z]	Poor service but good food.	bad terrible …
Named Entity Recognition	[X1] [X2] is a [Z] entity.	[X1] LREC-COLING is in Turin this year. [X2] Turin	organization location …
Summarization	[X] TL;DR: [Z]	Section 1: Course Details….	CS 521 is at 9:30 a.m. on Tuesdays and Thursdays in LCA 007….

Answer Search

- **Goal: Search for the highest-scoring text $\hat{\mathbf{z}}$ that maximizes $P(\mathbf{x}';\theta)$**
- First, define a set Z of possible answers $\mathbf{z}$
  - Might be a limited set of words (for classification) or the entirety of the language (for generation)
- Then, create **filled prompts** for each $\mathbf{z}$ by filling the intermediate answer slot in $\mathbf{x}'$ with that $\mathbf{z}$
  - $\mathbf{x}' = f_{\text{fill}}(\mathbf{x}',\mathbf{z})$
  - Note: If this results in a correct answer, we can consider this an **answered prompt**
- Finally, search over all $\mathbf{z} \in Z$ by calculating the probability of their corresponding filled prompts in the language model
  - $\hat{\mathbf{z}} = \operatorname{search}_{\mathbf{z} \in Z} P(f_{\text{fill}}(\mathbf{x}', \mathbf{z}); \theta)$

Answer Mapping

- Convert the highest-scoring intermediate answer $\hat{\mathbf{z}}$ to the output $\hat{\mathbf{y}}$
- In language generation tasks, this is trivial
  - The intermediate answer is the output!
- In classification tasks, a mapping needs to be defined between class labels and the potentially numerous intermediate answers that may lead to them



Now we understand how prompting works, but what should we think about as we design prompting methods?

- Design Considerations for Prompting
  - LLM choice
  - Prompt template engineering
  - Prompt answer engineering
  - Parameter training (how much, if any, is necessary?)

Prompt Template Engineering

- The process of creating a prompting function that results in the best performance on the downstream task
- May involve manual design, automated algorithms, or both
- What should be considered?
  - Prompt shape
  - Level of automation

Prompt Shape

- **Cloze prompt:** Fill-in-the-blank style
  - Well-suited to models pretrained with masked language modeling objectives
  - Good for classification tasks
- **Prefix prompt:** Continuation style
  - Well-suited to models pretrained with autoregressive generation objectives
  - Good for generation tasks
- Other shapes:
  - Text pair classification: Slots for at least two inputs

Level of Automation

- Manual Template Engineering
  - Templates that seem intuitive based on commonsense
  - Written by the researcher/model developer
  - Good starting point, but limited in several ways:
    - Time-consuming and requires high level of expertise
    - No guarantee that optimal prompts will be written
- Alternative: Automated Template Learning!

Automated Template Learning

Discrete vs. Continuous

- **Discrete Prompts:** Automatically learned prompt template is a text string
- **Continuous Prompts:** Automatically learned prompt is an embedding

Static vs. Dynamic

- **Static Prompts:** Same template for each input
- **Dynamic Prompts:** Custom template for each input

Discrete Prompting

- Also referred to as hard prompts
- Automatically search for prompts in discrete, natural language space
- Popular approaches:
  - Prompt Mining: Scrape a large text corpus for strings containing inputs and outputs, and then use the words or dependency paths between the inputs and outputs as templates
  - Prompt Paraphrasing: Create paraphrases of a seed prompt, and select the highest-scoring paraphrase on the training data
  - Gradient-Based Search: Search over the vocabulary to find sequences of tokens that trigger the underlying language model to generate the correct prediction
  - Prompt Scoring: Manually write candidate prompts and then choose the highest-scoring prompt for each input

Continuous Prompting

- Also referred to as soft prompts
- Intuition: Prompts aren't designed for human consumption; they're designed to elicit the right response from a language model …no need to use words!
- Thus, continuous prompts remove two constraints:
  - Embeddings no longer need to represent natural language words
  - Template no longer needs to be parameterized by the language model's parameters
- Instead, continuous prompts have their own tunable parameters that are optimized based on the downstream task

Example Continuous Prompting Strategies

Prefix Tuning	Tuning Initialized with Discrete Prompts	Hard-Soft Prompt Hybrid Tuning
Prepends a sequence of continuous task-specific vectors to the input	Initialize the search for an optimal continuous prompt using a discrete prompt	Insert tunable embeddings into a discrete prompt template

Prompt Answer Engineering

- Seeks to find an optimal mapping between an answer space and the discrete output labels
- Two key design considerations:
  - Answer shape
  - Answer design method

Answer Shape

- Common granularities:
  - **Token:** A token in the language model's vocabulary
  - **Span:** A short multi-token span
    - Often used with cloze prompts
  - **Sentence:** A sentence (or more generally, a longer document)
    - Often used with prefix prompts

How to choose an answer shape?

- Largely depends on the task
  - Token or span answer spaces → Classification or information extraction tasks
  - Long answer spaces → Language generation tasks

Answer Space Design Methods

- Similarly to prompt template design, can be manual or automated
- Automated techniques may include:
  - Discrete answer search
  - Continuous answer search

Manual Answer Space Design

- Space of potential answers and their mapping to the label space are manually defined by the researcher or model developer
- Numerous strategies can be taken:
  - Unconstrained Spaces: Answer space is the space of all tokens, fixed-length spans, or token sequences, and answers from this space are directly mapped to labels in the label space
  - Constrained Spaces: Space is constrained according to task characteristics (e.g., sets of words associated with topics relevant to classes), and those answers are mapped to the label space

Discrete Answer Search

- Similarly to manual prompt template engineering, manual answer spaces may not optimally map answers to labels
- Approaches to automatically search for the ideal mapping include:
  - Answer Paraphrasing:
    - Start with an initial answer space
    - Expand it using paraphrasing techniques
    - Define probabilities for labels based on the marginal probabilities of all answers in a paraphrased answer set
  - Prune-then-Search:
    - Generate an initial pruned answer space of several plausible answers
    - Search over this space to find the final set of answers
    - Select the best (highest probability) label for a particular answer

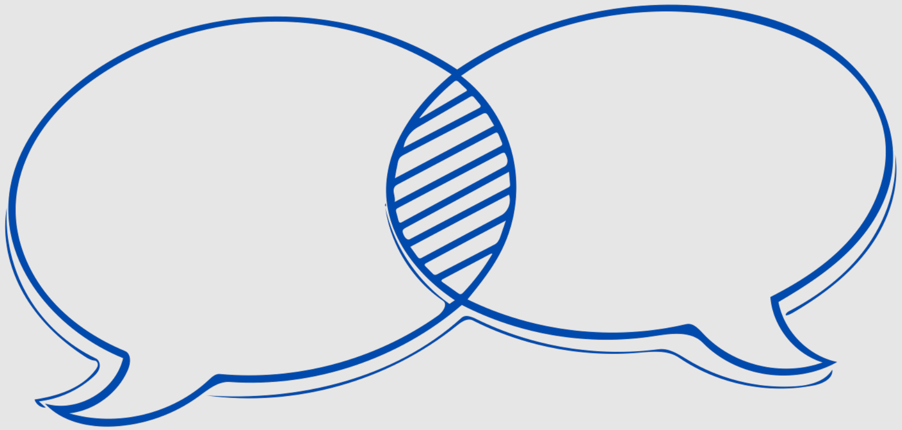
Continuous Answer Search

- Soft answer tokens can also be optimized alongside soft prompts
- In this case, embeddings are learned from scratch for each label

We know the basics of prompt design ...where can we go from here?

- **Multi-Prompt Learning:** No need to limit yourself to using one prompt!
- Strategies leveraging multiple prompts often achieve better performance
- Numerous approaches for multi-prompt learning, including:
  - Prompt ensembling
  - Prompt augmentation
  - Prompt composition
  - Prompt decomposition

Prompt Ensembling



- Use multiple unanswered prompts at inference time to make predictions
  - Works with both discrete and continuous prompts
- Advantages of prompt ensembling:
  - Leverages complementary advantages of multiple prompts
  - Alleviates the cost of prompt engineering (no need to spend time selecting a single best prompt!)
  - Stabilizes performance on downstream tasks

Prompt Ensembling Techniques

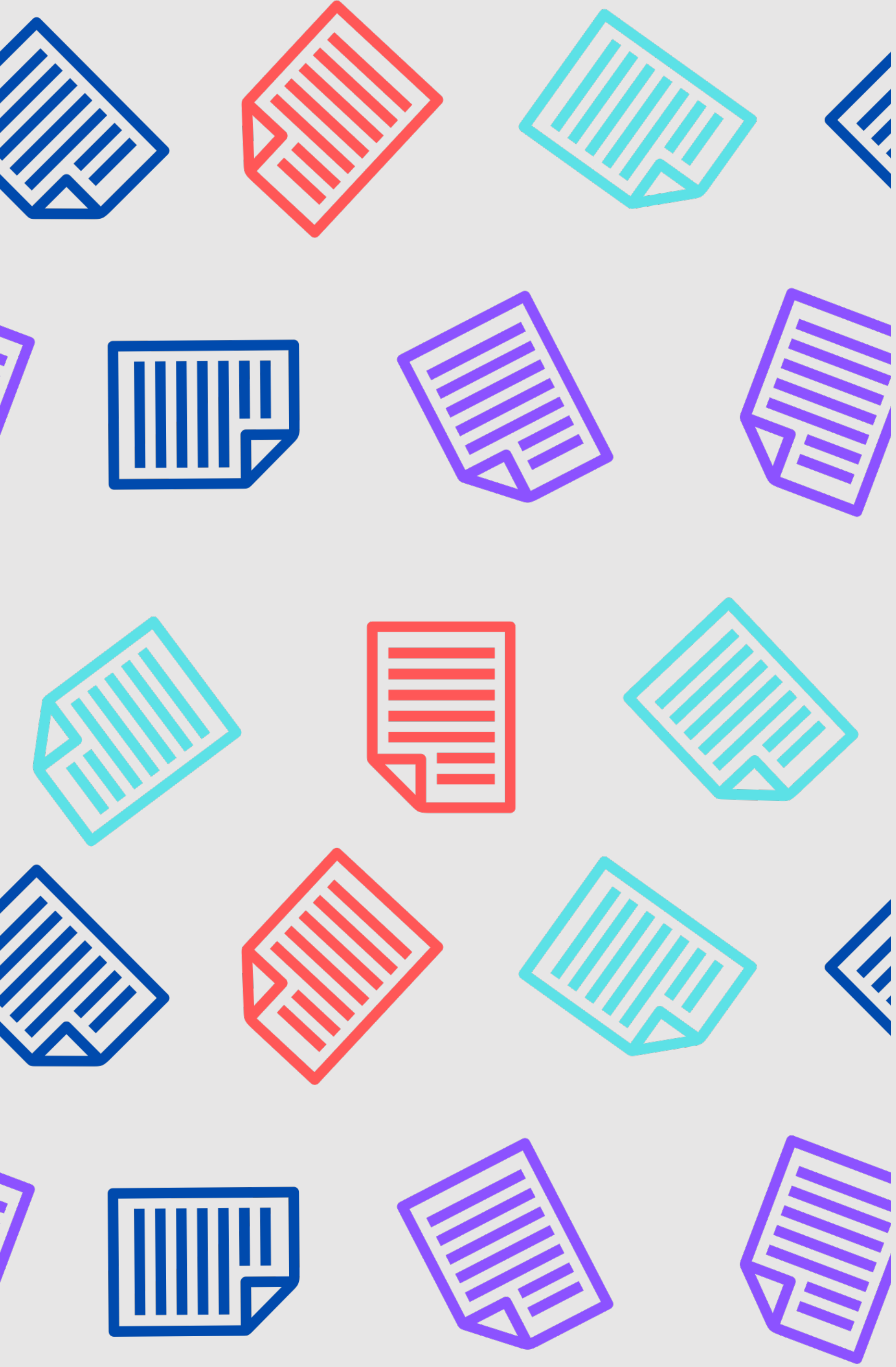
Based on earlier approaches from ensemble models in machine learning

Popular techniques include:

Uniform Averaging	Weighted Averaging	Majority Voting	Knowledge Distillation	Prompt Ensembling for Text Generation
• Average the answer probabilities across multiple prompts	• Compute a weighted average of answer probabilities across multiple prompts, where weights are associated with prompts based on prompt performance or optimized using a training set	• Take a majority vote for answers across multiple prompts	• Train a final model to distill knowledge ensembled across multiple prompts	• Generate output based on ensembled probability of the next word in the answer sequence, or score different decoded sequences and select the highest-scoring sequence

Prompt Augmentation

- Also referred to as demonstration learning
- Provides additional answered prompts to demonstrate to the language model how it should ideally answer to provided prompt
- Takes advantage of the language model's ability to learn patterns from only a few shots of training data
- Key considerations in prompt augmentation:
  - Sample selection
  - Sample ordering



Sample Selection

- The examples provided in prompt augmentation settings can influence performance!
- Common approaches:
  - Select samples that are **close to the input** in embedding space
  - Provide both **positive and negative** samples

Sample Ordering

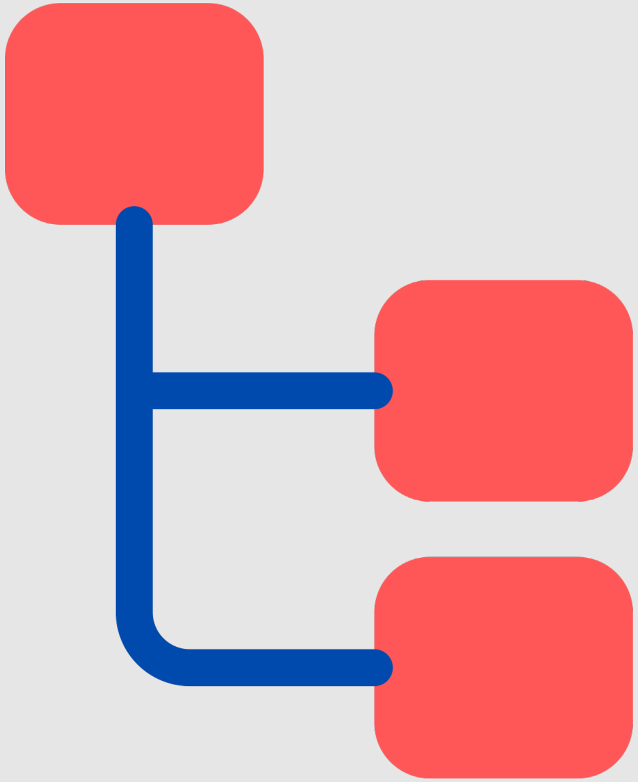
- The order of examples provided in prompt augmentation settings can also influence performance
- Common approaches:
  - Score different permutations of samples using entropy-based methods
  - Search for ideal permutations of samples

Prompt Composition

Create subprompts for subtasks of the broader task	Subprompts may be manually or automatically constructed
Define a composite prompt based on those prompts	Composition may use logic rules or other strategies to compose the subprompts into a single prompt

Prompt Decomposition

- Break the prompt itself into separate subprompts and have the language model answer each subprompt separately
- Useful for settings where numerous answers are needed for a single input sample (e.g., sequence labeling)
  - Requires that the input is first converted into text spans



Chain-of-Thought Prompting

- Designed to improve the reasoning abilities of LLMs by enabling them to decompose multi-step problems into intermediate steps
- How does this differ from other forms of prompting?
  - The model is prompted to give intermediate reasoning steps *before* providing its answer to a problem

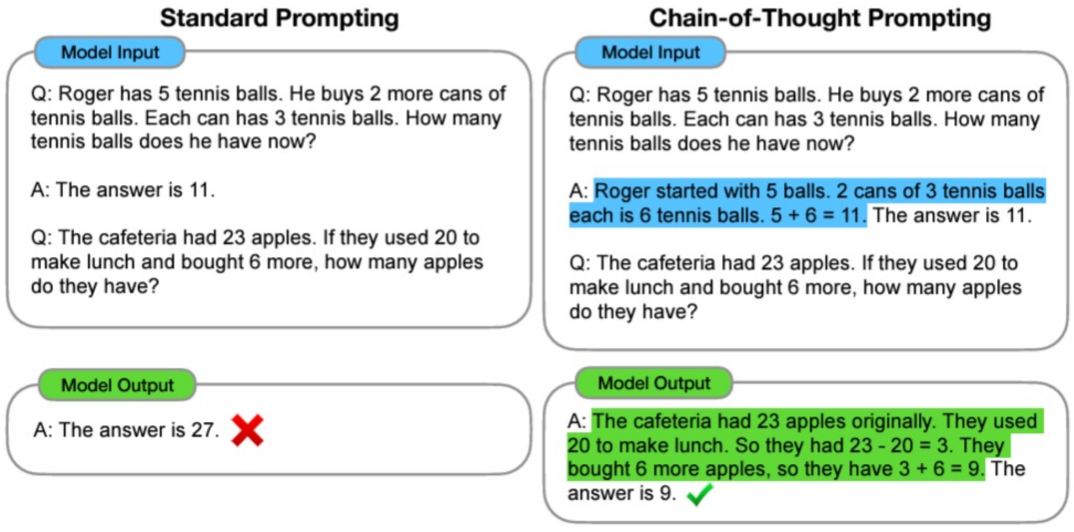


Figure 1: Chain-of-thought prompting enables large language models to tackle complex arithmetic, commonsense, and symbolic reasoning tasks. Chain-of-thought reasoning processes are highlighted.

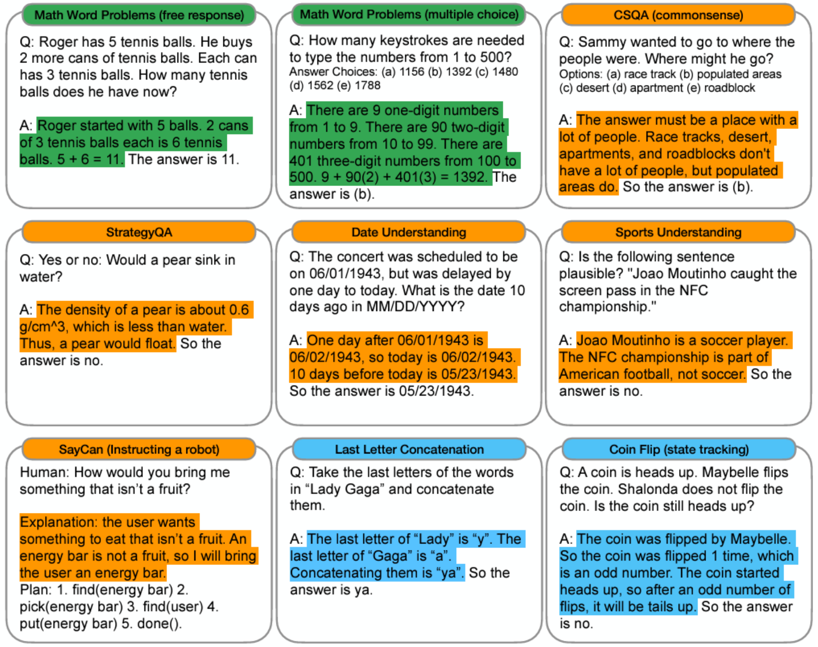


Figure 3: Examples of ⟨input, chain of thought, output⟩ triples for arithmetic, commonsense, and symbolic reasoning benchmarks. Chains of thought are highlighted. Full prompts in Appendix G.

Chain-of-Thought Prompting Elicits Reasoning in Large Language Models: https://openreview.net/pdf?id=_VjQlMeSB_J

Chain-of-Thought Prompting

- Advantages:
  - Intermediate steps can be solved individually (and perhaps more easily) than the overall problem
  - Applicable to any language-based task
- Disadvantages:
  - Usefulness seems to scale with LLM size (may not work overly well with smaller LLMs)

Interesting property of CoT prompting: Can be implemented for zero-shot settings simply by including "Let's think step by step" in the prompt!

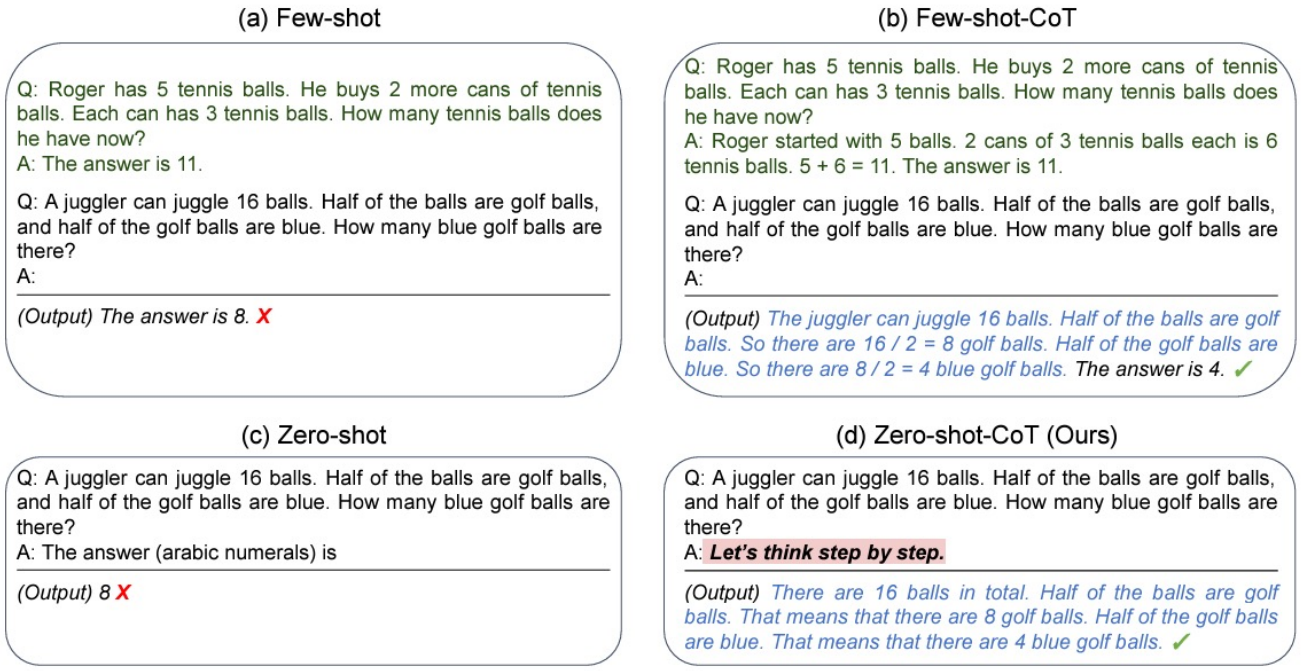


Figure 1: Example inputs and outputs of GPT-3 with (a) standard Few-shot ([Brown et al., 2020]), (b) Few-shot-CoT ([Wei et al., 2022]), (c) standard Zero-shot, and (d) ours (Zero-shot-CoT). Similar to Few-shot-CoT, Zero-shot-CoT facilitates multi-step reasoning (blue text) and reach correct answer where standard prompting fails. Unlike Few-shot-CoT using step-by-step reasoning examples **per task**, ours does not need any examples and just uses the same prompt "Let's think step by step" *across all tasks* (arithmetic, symbolic, commonsense, and other logical reasoning tasks).

Large Language Models are Zero-Shot Reasoners: https://openreview.net/pdf?id=e2TBb5y0yFf

Prompt-Aware Training

- Although it's possible to prompt models out of the box, it's also possible to train models in concert with prompting methods
- This may be achieved using a variety of training and parameter update approaches

Prompt-Aware Training Settings

- **Zero-Shot:** No training data is provided for the task of interest, and the LLM is used as-is to predict output
- **Few-Shot:** A few samples are available for training
- **Full-Data:** Many samples are available for training

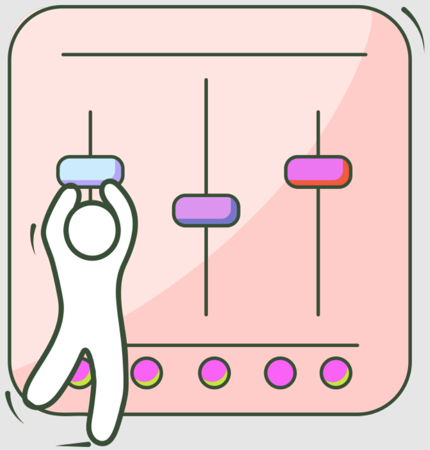
Prompting is more relevant to zero-shot or few-shot settings, since there generally aren't sufficient resources in those settings for the LLM to fully learn the downstream task.

What if training samples are used to help design prompts?

- Not true zero-shot learning (training samples are “seen” in a sense)
- However, this is distinct from model training/fine-tuning
    - It’s possible to use training samples to help author prompts without updating model weights in the process

Parameter Update Methods

- What are possible sources of parameters in prompt-based learning?
  - LLM
  - Prompts
- Different training categories exist depending on which of these sources are included in parameter updates



Promptless Fine-Tuning

- "Classic" transfer learning
  - Fine-tune a pretrained model to perform the downstream task, irrespective of the prompts that may be used with it
- Advantages:
  - Simple, powerful, and widely used
  - Model can more fully fit to larger training sets
- Disadvantages:
  - May overfit or have poor stability on smaller training sets
  - May forget information learned prior to fine-tuning (catastrophic forgetting)

Tuning-Free Prompting

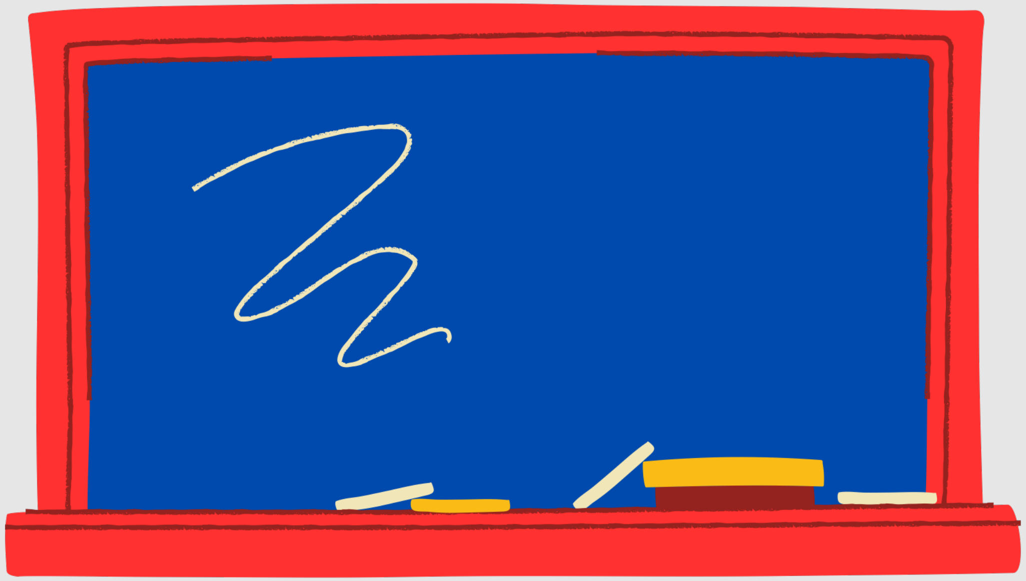
- Most straightforward version of prompting
  - Prompt the LLM directly, without updating any weights in the LLM based on the prompting method
- If the input is augmented with answered prompts, this form of prompting is referred to as **in-context learning (ICL)**
- Advantages:
  - Efficient, and avoids catastrophic forgetting in the language model
- Disadvantages:
  - Requires heavy prompt engineering
  - In in-context learning settings, including many answered prompts can slow inference

Fixed-LM Prompt Tuning

- Prompt-relevant parameters are added and fine-tuned based on downstream task samples, while the LLM remains frozen
- Advantages:
  - Often better performance than tuning-free prompting, while also retaining other advantages of that setting
- Disadvantages:
  - Requires prompt engineering, and resulting prompts are often not easy for humans to interpret or manipulate

Fixed-Prompt LM Tuning

- Instruction tuning
  - Fine-tunes the LLM using prompts with fixed parameters
- Advantages:
  - Efficient and more fully specifies the task to the model
- Disadvantages:
  - Requires template and/or answer engineering of some form, and may not generalize across downstream tasks



Prompt + LM Tuning

- Tunable prompt parameters are tuned alongside some or all of the parameters of the LLM
- Advantages:
  - Most expressive setting, and well-suited for large datasets
- Disadvantages:
  - Requires more compute resources, and may overfit with small datasets

What kind of problems can we solve with prompt-based learning?

- **Knowledge Probing:** How much (and what kind of) knowledge does the LLM have?
  - Factual Probing
  - Linguistic Probing
- **Structure Prediction:** What structured meaning representations can be extracted by the LLM?
  - Semantic Parsing
- **Classification:** How well can the task be reformulated and specified as a prompt?
- **Information Extraction:** What special units or relations can be extracted by the LLM?
  - Relation Extraction
  - Named Entity Recognition
- **Reasoning:** What do LLMs really understand about the world?
  - Commonsense Reasoning
  - Mathematical Reasoning
- **Question Answering:** To what extent can the LLM answer the specified question?
- **Text Generation:** How well does the LLM generate text, conditioned on the provided information?
- **Automated Evaluation of Text Generation:** Can prompts be used to evaluate the quality of generated text?

Prompting can also support other NLP tasks!

- Domain Adaptation
- Debiasing
- Dataset Construction
  - Generate datasets conforming to the provided instructions

What about multimodal prompts?

- It's possible to prompt LLMs with images:
  - Represent images as embeddings
  - Prompt LLMs with these embeddings
- Helpful to include demonstrations in this setting, especially if keeping the LLM frozen

How does prompt-based learning connect to other NLP and deep learning concepts?

- **Prompt Ensembling** → Facilitates ensemble learning without requiring that multiple models are trained
- **Prompt Augmentation** → Advances study of few-shot learning and larger-context learning
- **Discrete Prompt Search** → Borrows techniques from query reformulation
- **Continuous Prompt Fine-Tuning** → Adds control signals (such as those used in controllable text generation) to the prompts

Ongoing Challenges in Prompt-Based Learning

- Prompt Design
  - Creating prompts for classification and generation tasks is straightforward, but how should we design them when the end goal is language analysis?
- Prompt Answer Engineering
  - How should we engineer answer spaces with many classes, or with multi-token answers?
- Model Development
  - How should we select the base LLM, or the tuning strategy (if any)?
- Multi-Prompt Learning
  - What are the best prompts to include when performing prompt ensembling?
  - What is the best way to select demonstration samples, and how should they be ordered?

We also need to more fully understand:

- Theoretical and empirical analyses and guarantees for prompt-based learning
  - Just how much does prompt-based learning help a task?
- Optimal pretraining strategies for prompt-based learning

Prompting Resources

- OpenPrompt:
  - https://thunlp.github.io/OpenPrompt/
- PromptBench:
  - https://github.com/microsoft/promptbench
- Prompt Engineering Guide:
  - https://www.promptingguide.ai/
- Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing
  - https://dl.acm.org/doi/full/10.1145/3560815

Summary: Prompt-Based Learning

- **Prompts** are templates designed to encourage language models to produce desired task output when they are used as input
- Prompt-based learning involves determining the best:
  - Prompt design
  - Answer space
  - Underlying LLM
  - Parameter updates
- **Hard prompts** are discrete, natural language text, whereas **soft prompts** are embeddings
- **In-context learning** is a form of prompt augmentation in which demonstration samples are added to the prompt
- **Chain-of-thought prompting** is a form of prompting that asks the language model to break the broader question down into intermediate subproblems
- LLMs can be prompted directly without any fine-tuning, or they can be fine-tuned based on (and perhaps in concert with) the prompts with which they are used