Prompting and Instruction Fine-Tuning

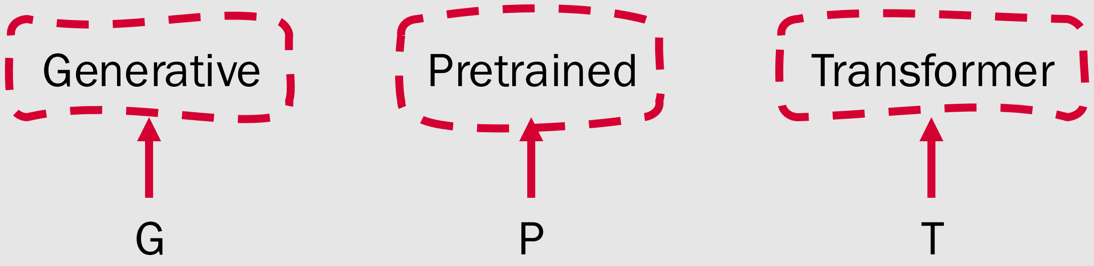
Natalie Parde, Ph.D.

Department of Computer Science

University of Illinois Chicago

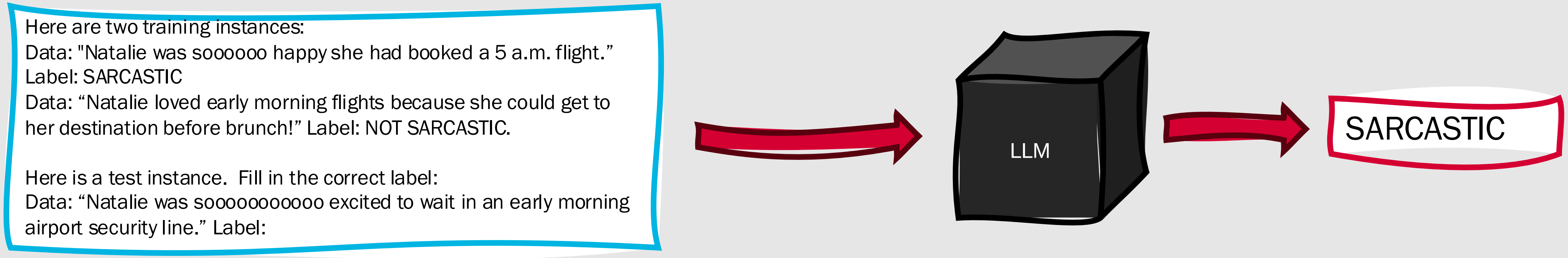
What’s most popular right now?

- 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



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 optionally fine-tune them and then prompt them to produce the correct labels or output for new tasks



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 (e.g., next word prediction)
- 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: A 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

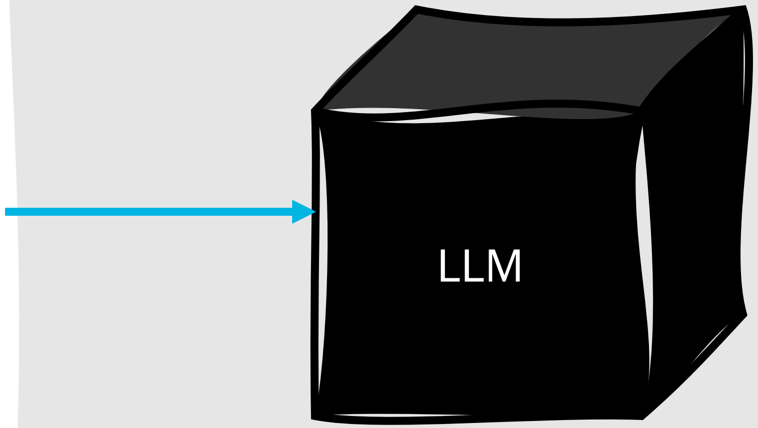
What does this look like?

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:

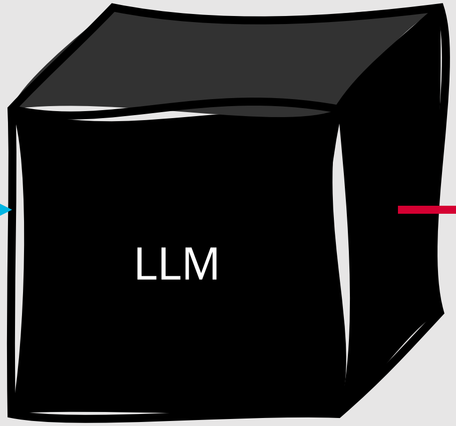
Method: an approach to interpret metaphors by mapping them to conceptual metaphor clusters

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

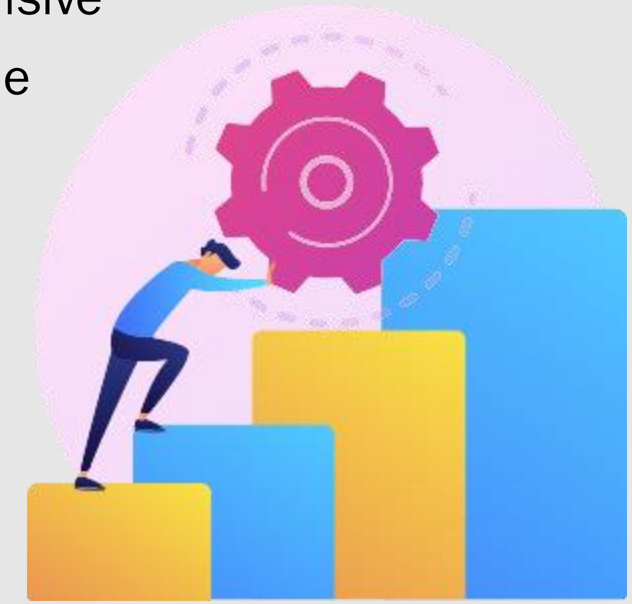
What are the advantages to including these tuning steps?

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

Limitations of Supervised Fine-Tuning

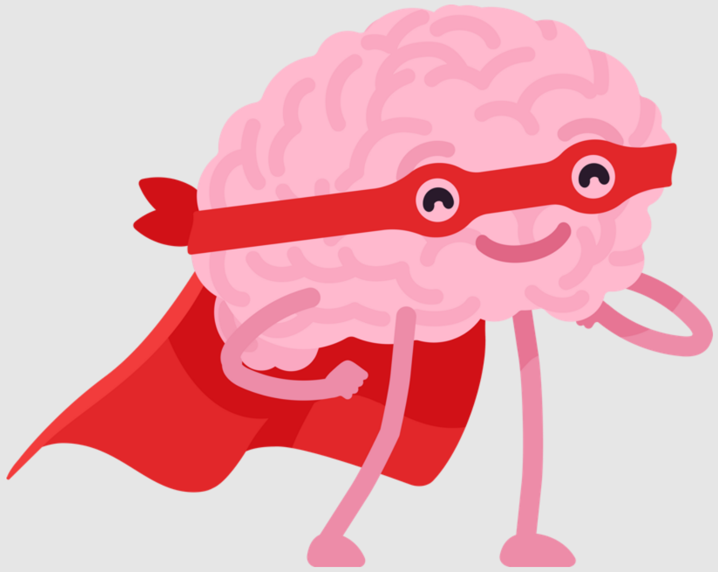
- There are many ways to complete a prefix by performing the task specified in the prefix, but some of those ways are much better than others
- Instruction tuning datasets provide demonstrations regarding how to perform the task specified in the prefix, but no ratings for those demonstrations!

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

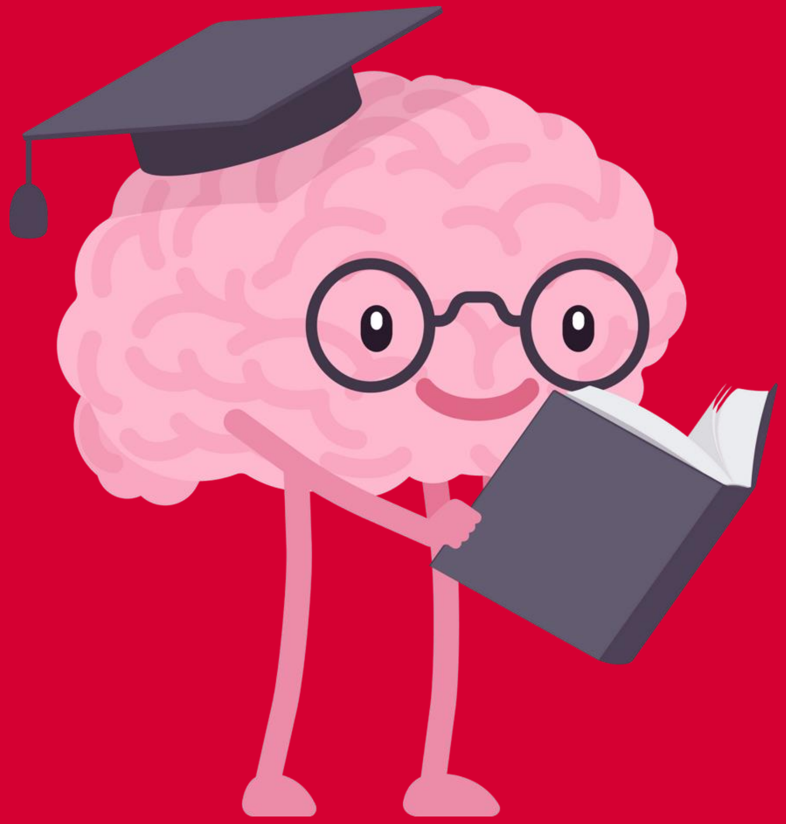
Prompts are randomly selected from a distribution → These prompts are passed into an LLM → Output from the LLM is scored using a reward model

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

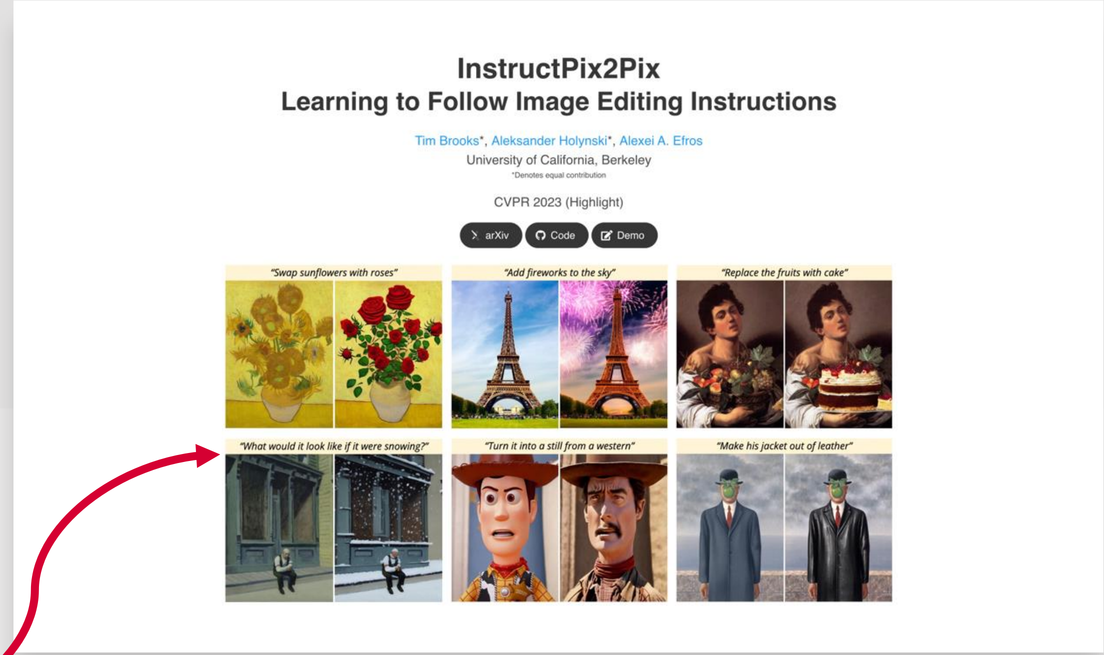
- RLHF has only very recently been applied to language modeling setups, and much is still unknown!
- Big issue currently: Hallucination
  - LLMs tend to make up information (and phrase misinformation confidently!)
- Why does this happen?
  - Two leading hypotheses:
    - LLMs don't understand the causes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

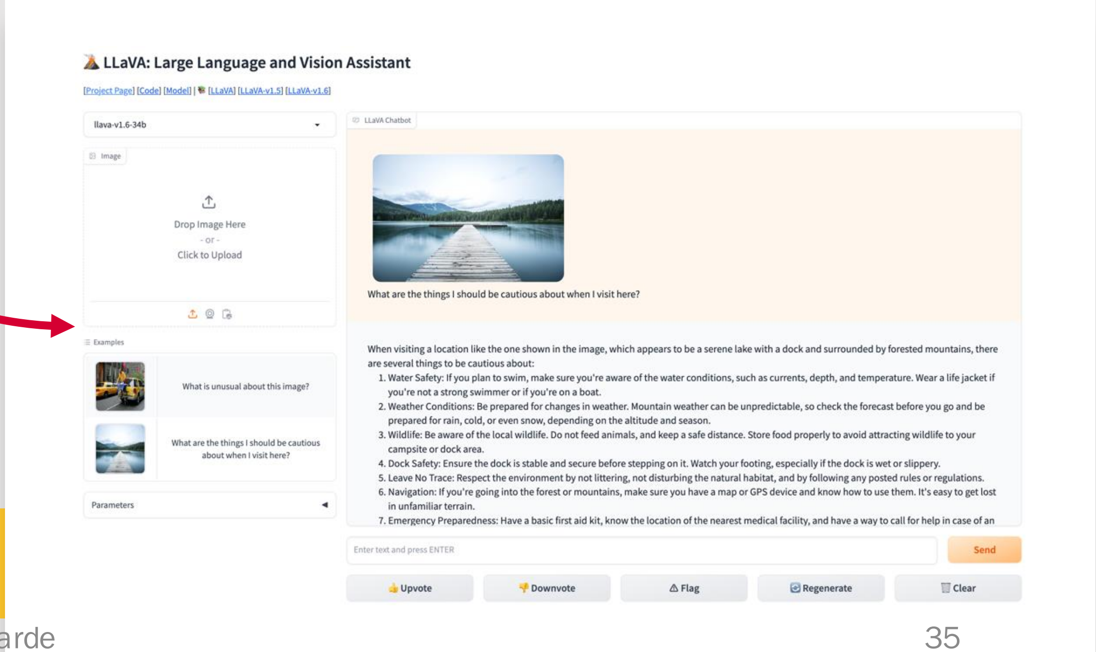
We’ve trained our model using supervised fine-tuning and RLHF …now what?

- 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

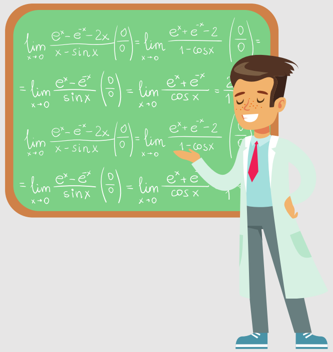
- Typically involves extra steps
  - **InstructPix2Pix:** First fine-tunes an LLM to predict (Image Caption, Edit Instruction, Updated Image Caption) triples, and then uses a text-to-image model to convert (Image Caption, Updated Image Caption) pairs to image pairs
    - https://github.com/timothybrooks/instruct-pix2pix
  - **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

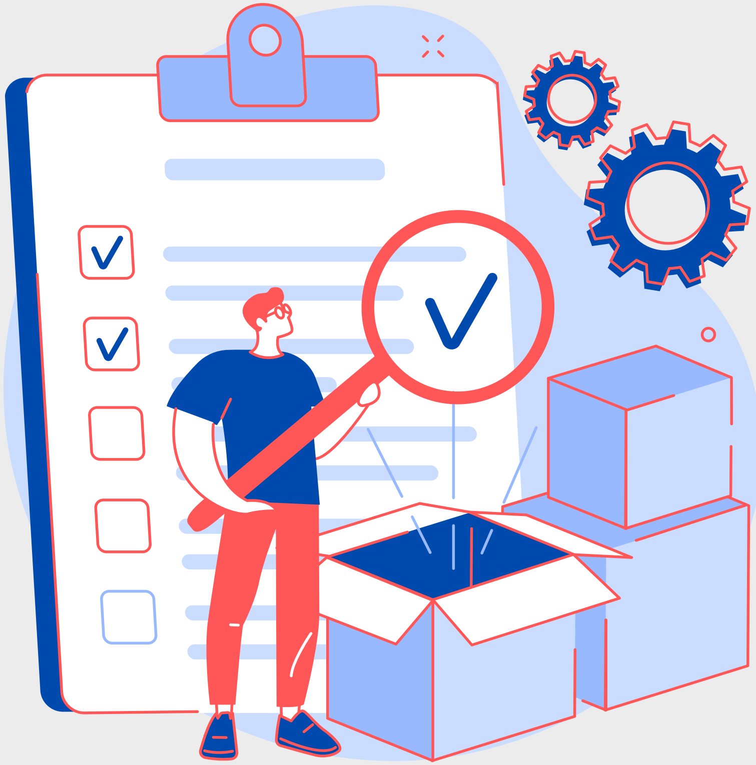
- Alpha: Scaling factor
  - Determines the extent to which the model is adapted to new data, by adjusting the contribution of the update matrices during training
  - Lower values → Maintain existing knowledge more than higher values
- Target Modules: Specific weights and matrices to be trained
  - May 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
- Difficult to evaluate instruction-tuned models in a standardized way!
  - 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

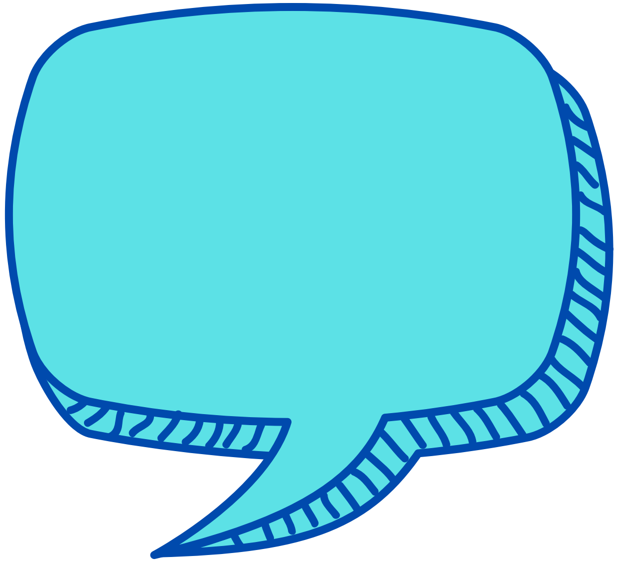
- Axolotl: https://github.com/OpenAccess-AI-Collective/axolotl
- LLaMA-Factory: https://github.com/hiyouga/LLaMA-Factory
- Open-Instruct: https://github.com/allenai/open-instruct

- 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

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?

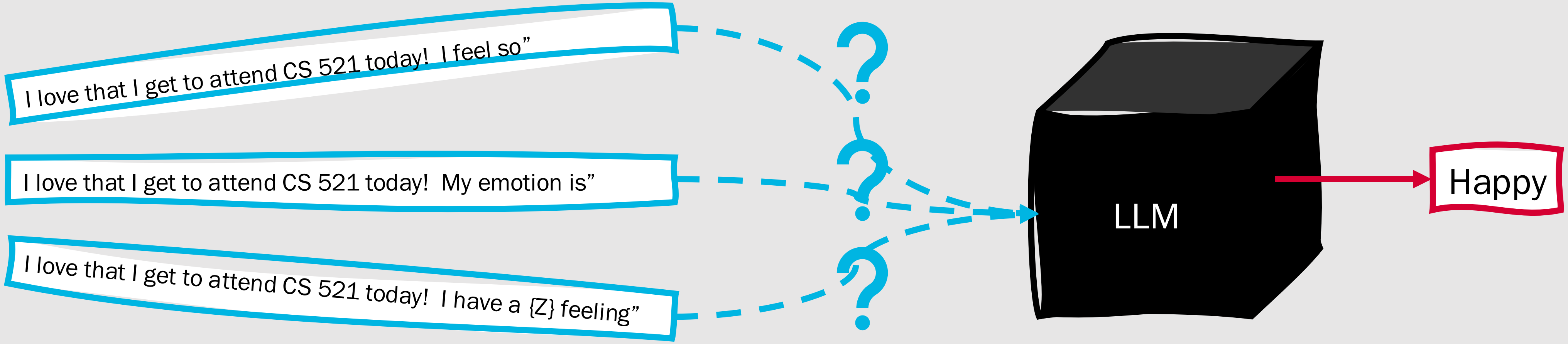
- Take a large model that has already been trained to model language
- Develop **prompt templates** for your task
  - Prompt templates can be manually or automatically constructed
- Develop an approach for **answer engineering**
  - Build an answer space (set of possible answers that your model may generate)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

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

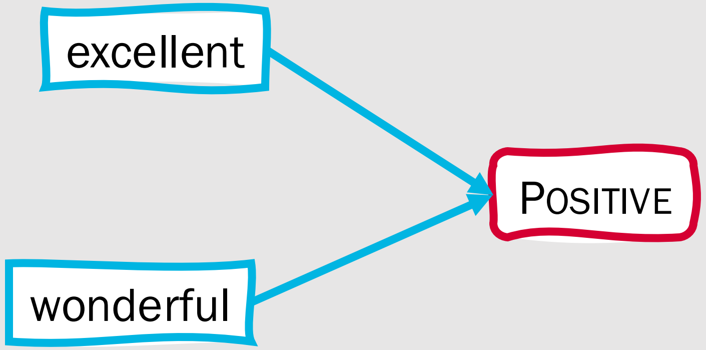
TASK	TEMPLATE	INPUT [X]	ANSWER [Z]
Topic Identification	[X] The text is about [Z]	She carefully read the journal article.	sports science …
Aspect-Based Sentiment Analysis	[X] How is the service? [Z]	Poor service but good food.	bad terrible …
Named Entity Recognition	[X1] [X2] is a [Z] entity.	[X1] LREC-COLING 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 $\hat{z}$ to the output $\hat{y}$
- 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

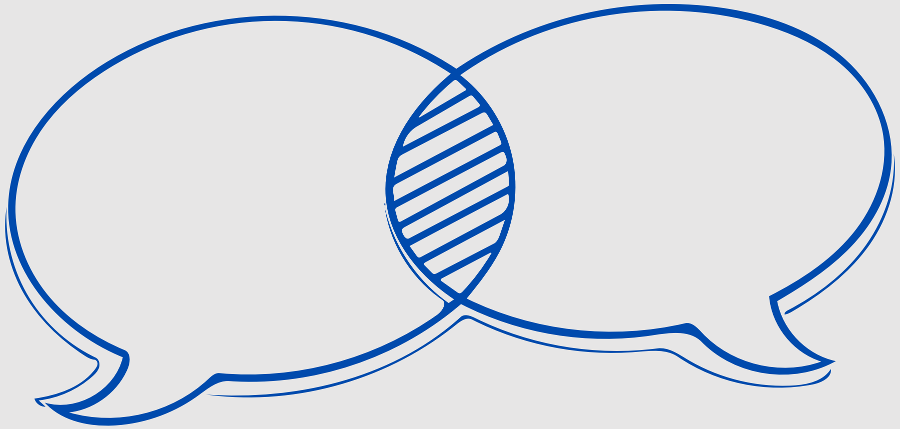
- Soft answer tokens can also be optimized alongside soft prompts
- In this case, embeddings are learned from scratch for each label

Summary: Instruction Tuning and Prompting

- **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
- **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

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!
- 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!)

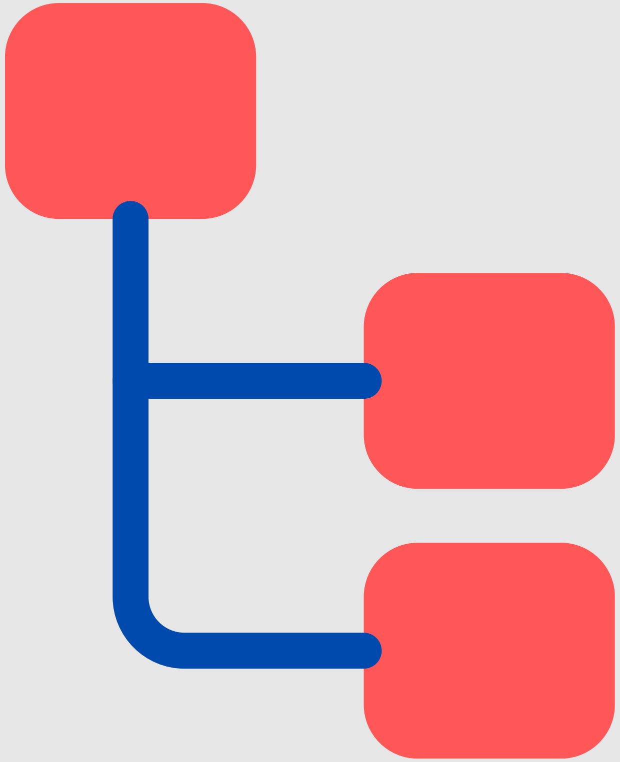
Uniform Averaging	Weighted Averaging	Majority Voting	Knowledge Distillation	Prompt Ensembling for Text Generation
• Average the answer probabilities across multiple prompts	• Compute a weighted average of answer probabilities across multiple prompts, where weights are associated with prompts based on 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 samples 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
    - Select samples that are close to the input in embedding space
    - Provide both positive and negative samples
  - Sample ordering
    - Score different permutations of samples using entropy-based methods
    - Search for ideal permutations of samples

Prompt Composition and Decomposition

- Prompt Composition
  - Create subprompts for subtasks of the broader task
  - Define a composite prompt based on those prompts
- 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)



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

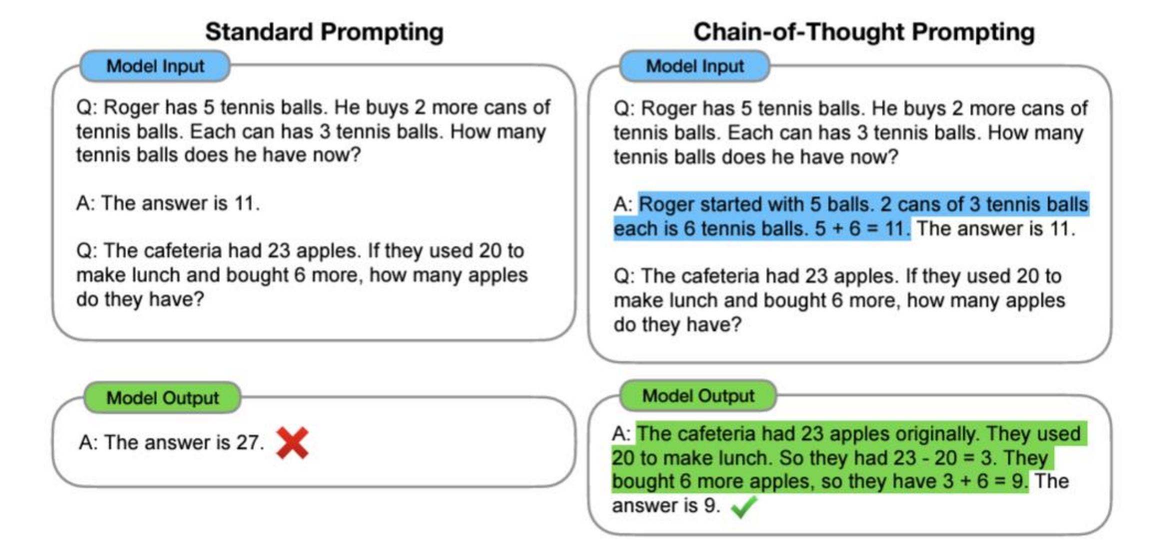


Figure 1: Chain-of-thought prompting enables large language models to tackle complex arithmetic, commonsense, and symbolic reasoning tasks. Chain-of-thought reasoning processes are highlighted.

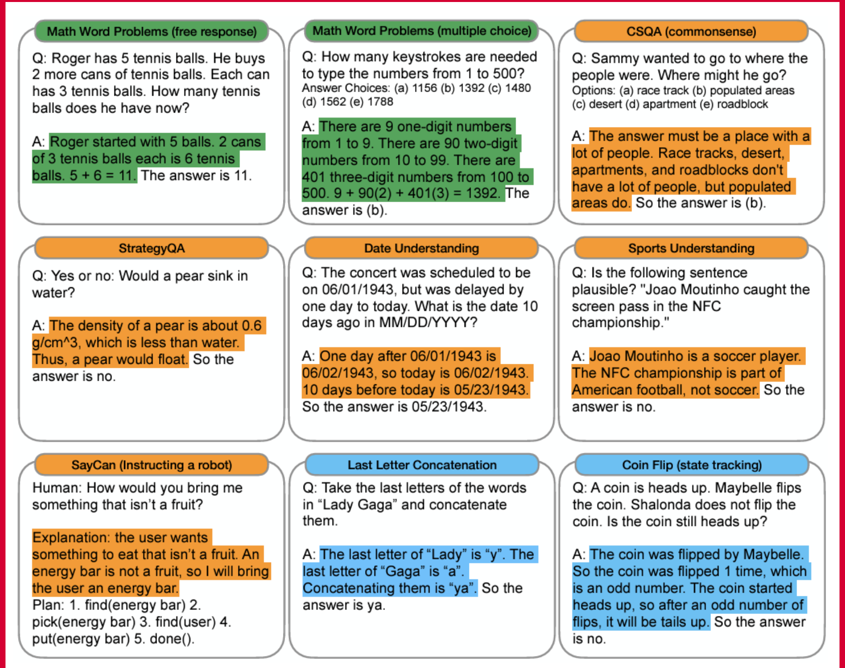


Figure 3: Examples of ⟨input, chain of thought, output⟩ triples for arithmetic, commonsense, and symbolic reasoning benchmarks. Chains of thought are highlighted. Full prompts in Appendix G.

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!

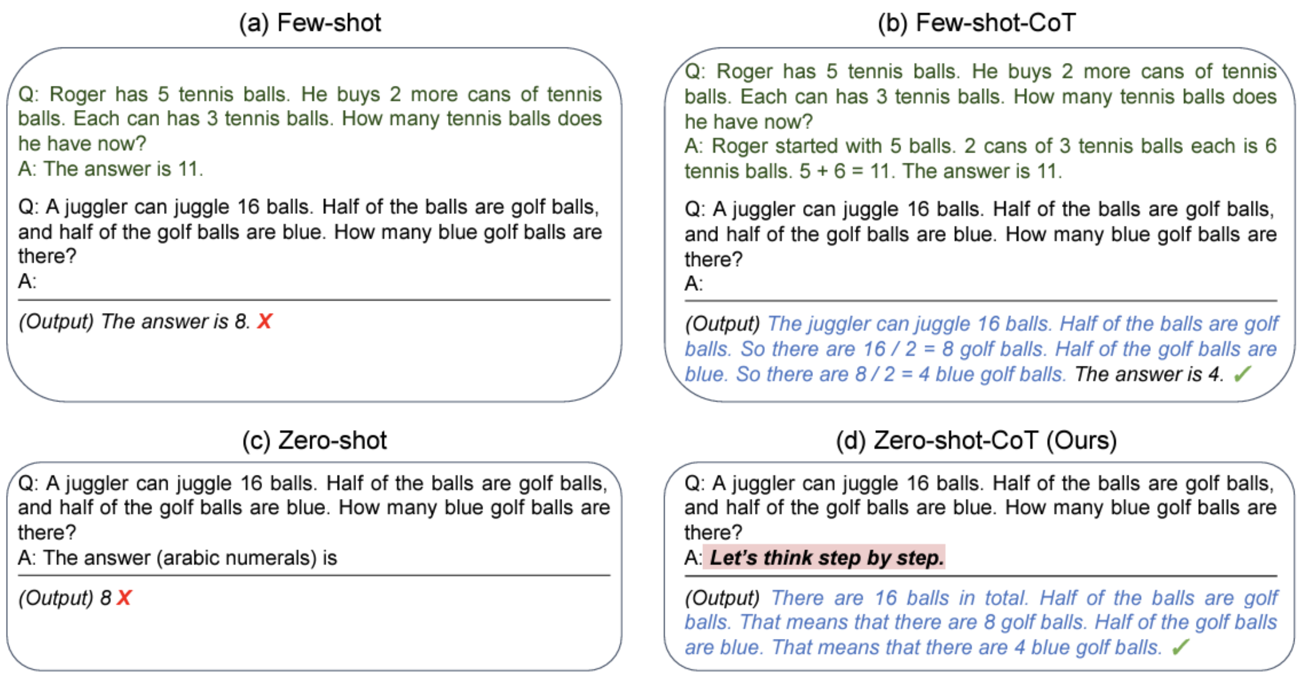


Figure 1: Example inputs and outputs of GPT-3 with (a) standard Few-shot ([Brown et al., 2020]), (b) Few-shot-CoT ([Wei et al., 2022]), (c) standard Zero-shot, and (d) ours (Zero-shot-CoT). Similar to Few-shot-CoT, Zero-shot-CoT facilitates multi-step reasoning (blue text) and reach correct answer where 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

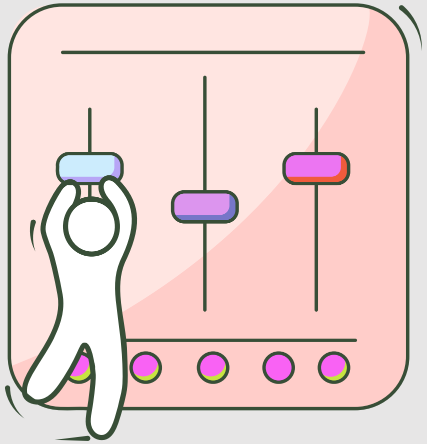
Prompt-Aware Training Settings

- **Zero-Shot:** No training data is provided for the task of interest, and the LLM is used as-is to predict output
  - What if training samples are used to help the developer design zero-shot prompts?
    - Not quite ”true” zero-shot, but no weight updates are involved.
- **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.

Parameter Update Methods

- Although it's possible to prompt models out of the box, it's also possible to train models in concert with prompting 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

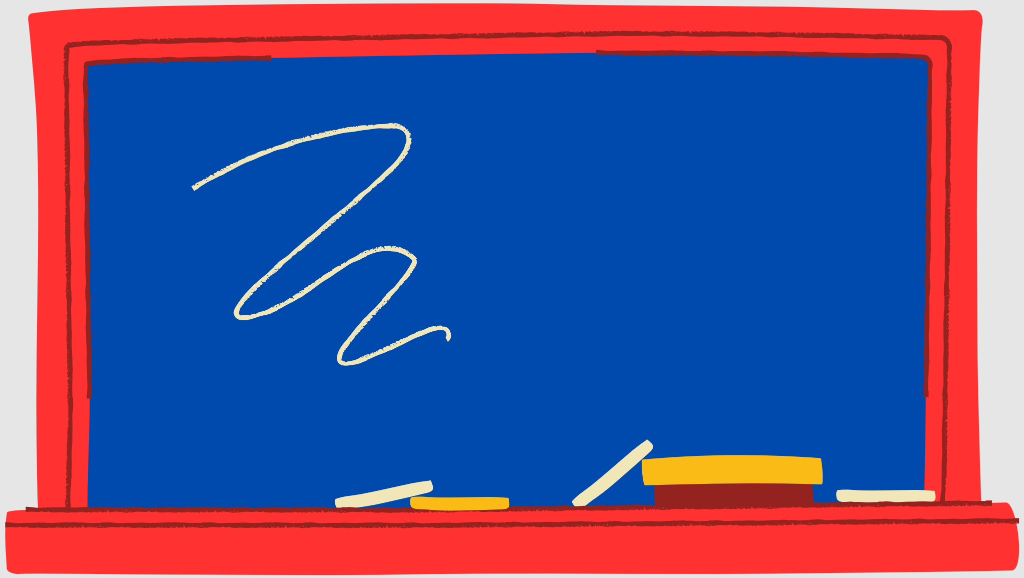
- Most straightforward version of prompting
  - Prompt the LLM directly, without updating any weights in the LLM based on the prompting method
- Advantages:
  - Efficient, and avoids catastrophic forgetting in the language model
- Disadvantages:
  - Requires heavy prompt engineering
  - Some in-context learning settings (i.e., prompt augmentation, or lots of context) 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?

Dataset Construction: Can we generate datasets conforming to the provided instructions?

Multimodal Tasks: Can we prompt LLMs with images?

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?
- Theoretical and empirical analyses and guarantees for prompt-based learning
- 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

Essentials of NLP Research

Natalie Parde

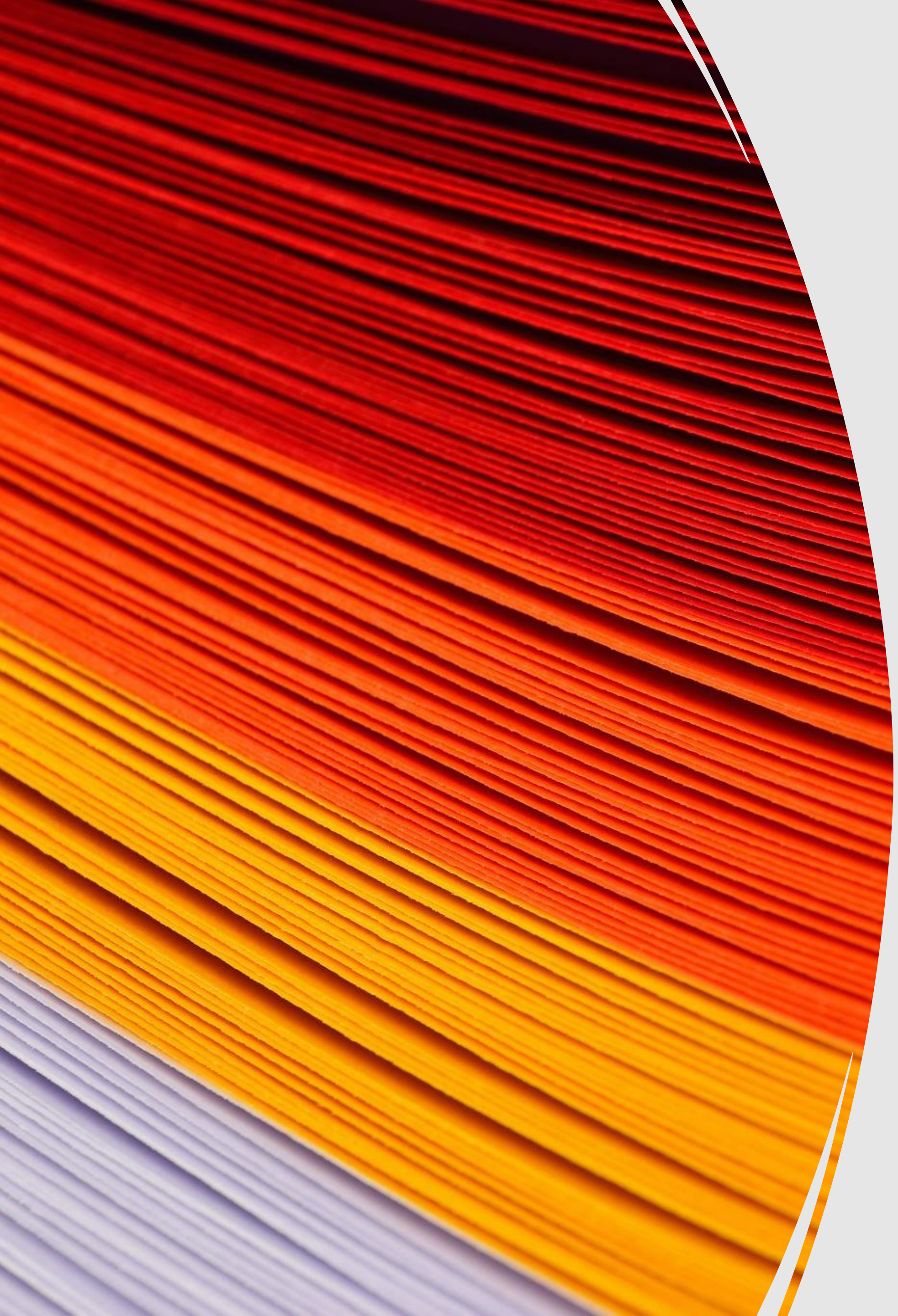
Literature Review

- As we've seen already in class, NLP is a diverse field with researchers pursuing many different problems
- It's also currently a very popular research field
  - Good for visibility, but makes literature review difficult!
- How do we perform good NLP literature reviews?

Submission Topics

ACL 2025 aims to have a broad technical program. Relevant topics for the conference include, but are not limited to, the following areas (in alphabetical order):

- Computational Social Science and Cultural Analytics
- Dialogue and Interactive Systems
- Discourse and Pragmatics
- Efficient/Low-Resource Methods for NLP
- Ethics, Bias, and Fairness
- Generation
- Human-Centered NLP
- Information Extraction
- Information Retrieval and Text Mining
- Interpretability and Analysis of Models for NLP
- Language Modeling
- Linguistic theories, Cognitive Modeling and Psycholinguistics
- Machine Learning for NLP
- Machine Translation
- Multilinguality and Language Diversity
- Multimodality and Language Grounding to Vision, Robotics and Beyond
- NLP Applications
- Phonology, Morphology and Word Segmentation
- Question Answering
- Resources and Evaluation
- Semantics: Lexical and Sentence-Level
- Sentiment Analysis, Stylistic Analysis, and Argument Mining
- Speech recognition, text-to-speech and spoken language understanding
- Summarization
- Syntax: Tagging, Chunking and Parsing
- **Special Theme**: Generalization of NLP Models



Primary NLP Paper Sources

- Most highly competitive NLP paper venues are conferences
  - Relatively common in CS, but very different from most other research fields
- A few notable journal exceptions:
  - Computational Linguistics: https://direct.mit.edu/coli
  - Transactions of the Association for Computational Linguistics: https://transacl.org/index.php/tacl
  - Computer Speech & Language: https://www.sciencedirect.com/journal/computer-speech-and-language

NLP Conferences, Loosely/Subjectively Ranked

Top Tier:

- Annual Meeting of the Association for Computational Linguistics (ACL)
- Conference on Empirical Methods in Natural Language Processing (EMNLP)
- Conference of the North American Chapter of the ACL (NAACL)

A bit easier to get accepted to, but also very well-regarded:

- International Conference on Computational Linguistics (COLING)
- Language Resources and Evaluation Conference (LREC)
- Conference of the European Chapter of the ACL (EACL)
- Conference of the Asian Chapter of the ACL (AACL)

Other Conferences that Often Have NLP Papers

Conference of the Association for the Advancement of Artificial Intelligence (AAAI)

International Joint Conference on Artificial Intelligence (IJCAI)

Conference on Neural Information Processing Systems (NeurIPS)

International Conference on Learning Representations (ICLR)

International Conference on Machine Learning (ICML)

What is “Findings of (EMNLP/ACL)”?

- New initiative that started several years ago, given the large increases in submissions to top NLP venues
- “Findings” papers have been determined to meet the soundness criteria for the conference, but weren’t selected for inclusion, usually because:
  - They focus on a problem that has less general applicability
  - They were a bit more borderline in their reviews
  - The organizers received too many high-quality submissions on the topic and needed to balance the program
- Bottom Line: Still good papers! Can be viewed similarly to COLING/EACL/etc. in terms of prestige

2.1 SIGARAB: SIG on Arabic Natural Language Processing
2.2 SIGANN: SIG on annotation
2.3 SIGBioMed: SIG on biomedical natural language processing
2.4 SIGDAT: SIG on linguistic data and corpus-based approaches to NLP
2.5 SIGDIAL: SIG on discourse and dialogue
2.6 SIGEDU: SIG on building educational applications
2.7 SIGEL: SIG on endangered languages
2.8 SIGFinTech: SIG on Economic and Financial Natural Language Processing
2.9 SIGFSM: SIG on finite state methods and models in natural language processing
2.10 SIGGEN: SIG on natural language generation
2.11 SIGHAN: SIG on Chinese language
2.12 SIGHUM: SIG on language technologies for the socio-economic sciences and the humanities
2.13 SIGLEX: SIG on the lexicon
2.14 SIGMOL: SIG on mathematics of language
2.15 SIGMORPHON: SIG for computational morphology, phonology, and phonetics
2.16 SIGMT: SIG on machine translation
2.17 SIGNLL: SIG on natural language learning
2.18 SIGPARSE: SIG on parsing
2.19 SIGREP: SIG on representation learning
2.20 SIGSEA: SIG on Southeast Asian Natural Language Processing
2.21 SIGSEC: SIG on NLP security
2.22 SIGSEM: SIG on computational semantics
2.23 SIGSLAV: SIG on Slavic natural language processing
2.24 SIGSLPAT: SIG on speech and language processing for assistive technologies
2.25 SIGSLT: SIG on Spoken Language Translation
2.26 SIGSUMM: SIG on Summarization
2.27 SIGTURK: SIG on Turkic Languages
2.28 SIGTYP: SIG on Linguistic Typology
2.29 SIGUR: SIG on Uralic languages
2.30 SIGWAC: SIG on web as corpus
2.31 SIGWrit: SIG on writing systems and written language

What about workshops?

- The NLP community has a wide range of workshops, each tailored to specific subproblems
- Workshops are often hosted by special interest groups (SIGs) of the ACL
- Workshop papers often describe more preliminary work, to be discussed among a smaller, focused group of attendees

Information overload 🤯 how do I choose which papers to read?

- The “curse” of being a popular field: new papers are always coming out, and it’s nearly impossible to keep up with all of them!
- My research group usually follows an approximation (adapted a bit for the CS context) of the **PRISMA systematic review process**: https://www.prisma-statement.org/

Steps for Conducting a Systematic Literature Review (Part 1)

- Decide on the specific keywords you'll use for your search
  - Necessary for limiting your search (otherwise you may never stop searching!) and helpful for justifying literature review parameters to paper reviewers
- Select your source paper databases/venues
  - Most NLP papers are available through the ACL Anthology: https://aclanthology.org/
  - Other databases that commonly host NLP papers are the ACM Digital Library: https://dl.acm.org/ and IEEE Xplore: https://ieeexplore.ieee.org/Xplore/home.jsp
  - Generally better to avoid including arXiv
    - Hard to verify the quality of papers (arXiv papers that are peer-reviewed will also show up in your results from the other databases)
    - Huge volume of papers
  - Okay to use Google Scholar, but likely to also return irrelevant search results (expect to need to filter liberally)

Steps for Conducting a Systematic Literature Review (Part 2)

- **Run your search query** in your selected database(s)
- From among the retrieved papers, **remove duplicates** (keep track of how many duplicates were removed)
- **Establish inclusion/exclusion criteria** for your review
  - Common inclusion criteria in my research group: Paper is written in English, paper focuses on technical method, paper focuses on topic X
  - Common exclusion criteria in my research group: Paper is an abstract, paper is not publicly available

Steps for Conducting a Systematic Literature Review (Part 3)

- For all remaining retrieved papers, **scan the titles** and remove papers that don't meet the inclusion/exclusion criteria (keep track of how many papers are removed)
- For all papers that passed the title scan, **read the abstracts** and remove papers that don't meet the inclusion/exclusion criteria (keep track of how many papers are removed)
  - This should leave you with a much more manageable set of papers, while ensuring you've comprehensively covered the literature!
- For all remaining papers, **create a spreadsheet** where you can log the paper title, citation, and attributes of interest as you read them

Research Methods

- After completing your literature review, you should have a good understanding of the current state of your research area and the gaps that remain!
- At this point, you can begin narrowing down your specific research question
- What makes a good research question?
  - Should be **specific** enough to answer
  - Should be **grounded** in existing evidence
  - Should be mapped to one or more testable **hypotheses**

Example Research Questions

Good 😊

- Can prosodic features improve the performance of Transformer-based models for speech-based sentiment analysis?
- How can self-supervised learning be used to improve empathy detection in social media text?
- What are the limitations of benchmark datasets in evaluating cross-linguistic sarcasm detection?

Not So Good 😕

- How can NLP be used for healthcare?
  - Lacks a specific problem or method
- What is the best approach for sentiment analysis?
  - Vague and too difficult to answer comprehensively
- Does BERT perform better than an LSTM for named entity recognition?
  - Already well-studied
- How can we make an NLP model understand human emotions?
  - Too broad and ill-defined

Once you've defined your research question, you can think about how to answer it!

- How will you convincingly determine that the hypothesis linked to your research question is or is not correct?
  - Think about the experiments that are necessary to determine this, and what resources are needed for them!
- Important to think about:
  - Will you need to collect a new dataset?
  - Will you need access to expensive GPUs?
  - Will you need IRB approval?
  - What will your evaluation look like?

NLP Evaluation

- Make sure to compare your method to strong **baselines**
  - Existing highest-performing methods for the same problem, if there are any
  - If there are not, compare to generally high-performing approaches in NLP:
    - Prompting an LLM
    - Fine-tuning BERT
- Make sure to use commonly accepted or well-justified **evaluation metrics**
  - Often better to use a variety of evaluation metrics (e.g., precision, recall, and F_1, or BLEU and perplexity scores)

Make sure to also analyze your findings!

- Research doesn't end with a score from a metric: make sure to review your findings to understand why the approach did/didn't work
- This can be done via manual **error analysis**:
  - For all or a random sample of mispredictions, create a spreadsheet and go through the mispredictions one by one, making note of defining attributes (e.g., "contains many misspellings" or "short/ambiguous")
  - Calculate statistics across this spreadsheet to determine common error categories, and think about ways to address them
- Recently, some researchers have also explored automated approaches to this
  - For example, LLM-as-a-judge: https://dl.acm.org/doi/10.5555/3666122.3668142

Reproducibility

- When you conduct research, it's important to ensure that your findings are reproducible, both by yourself and by others interested in expanding upon your work
- Reproducible research should have:
  - Publicly available data and source code, unless prohibited by IRB or ethics guidelines
  - Clear documentation
  - Clearly specified hardware and library requirements
  - Clearly specified hyperparameters (including information about the random seed(s) used)

Statistical Significance

- One way to improve the reliability of your reported results is to calculate statistical significance
- Depending on the nature of your dataset and research problem, different statistical significance tests may be most appropriate: https://aclanthology.org/P18-1128.pdf

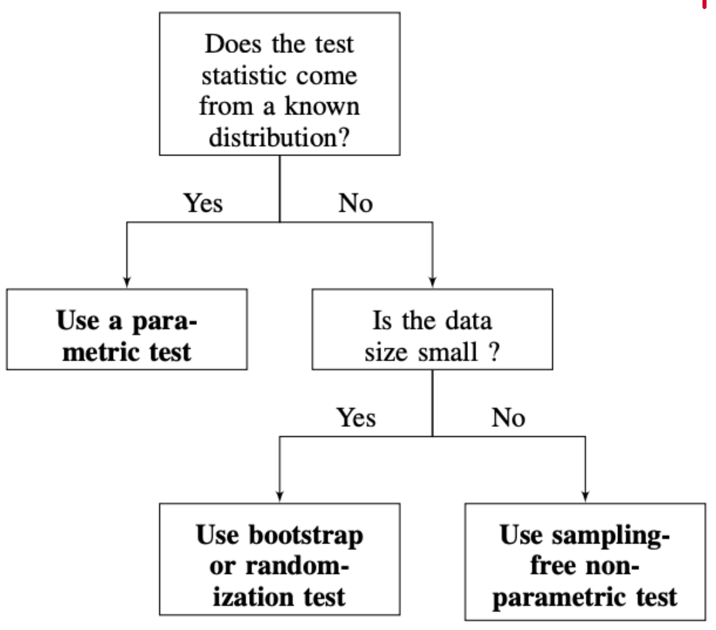


Figure 1: Decision tree for statistical significance test selection.

Averaging Results

- Even if you're not able to run full statistical significance tests, it's good to at minimum report **results averaged across multiple experimental runs**
- By calculating performance metrics across multiple runs, you can also report **standard deviation and/or variance**
  - This makes it easier for people reproducing your work to know if they've done something incorrectly, or if they're just falling within the expected range of performance

Research Communication

- You've completed your research study, analyzed the findings, and ensured that they're reproducible …time to **publicize** your results!
- There are two main ways you'll need to communicate your research findings:
    - **Written manuscripts**
    - **Oral presentations**

Strategies for Getting Started

Approach #1: Summarize research

- Do all of your research first, and then write about it
- Pro:
  - Allows you to guide your narrative by your experimental findings
- Con:
  - Can result in slower research (e.g., experiments that turn out to be unnecessary!)
  - You might forget some details along the way

Approach #2: Write and research simultaneously

- Draft your outline before starting your experiments, and fill in details as you go
- Pro:
  - Provides you with a clear roadmap for your research
- Con:
  - Can result in more stressful research (e.g., experiments don't turn out as planned!)
  - You might not have a good idea of which experiments are needed until you start your research

Getting Down to the Details

A “typical” CS research paper has several components

- Introduction
- Related work
- Methods
- Evaluation
- Discussion
- Conclusion

These components may vary depending on your project

- Separate “Data” section
- Separate “Results” section
- Discussion merged with conclusion

Introduction

Sets the tone

Provides the motivation for the research

Typically contains the following:

Problem statement: Broadly speaking, what problem is this research solving?	Summary: Briefly, what is this paper about?	Rationale: Why is this necessary or useful?	Contributions: What are the key takeaway points that set this work apart from others?

Related Work

- Situates your work in the broader research context
- Explains what has been done so far, and what the limitations are
  - What makes your research unique?
- Organizational tips:
  - Group relevant work based on underlying commonalities
  - For each paper (or group of related papers), provide a high-level summary and explain how your own work differs
- Make sure to cite all relevant papers!
  - Other researchers working on your topic are likely to be your reviewers

Methods

- Describes what you did and how you did it
- Try to include any details necessary for replicating your work:
  - What your data looks like and how it was collected (if not including a separate “Data” section)
  - How you implemented your approach(es)
    - External libraries
    - Model parameters
  - Why you made your design decisions
- Depending on your methods, it may help to supplement your writing with figures, algorithms, equations, or proofs

Evaluation

Explains how you assessed the quality of your work

Describes any:

- Baseline or alternative models or conditions
- Training settings, if relevant
  - Training/validation/test splits
  - Hyperparameter tuning
- Evaluation metrics

Usually includes results

- Objectively describe the results
- Use measures that make it easy to interpret the findings (e.g., X outperforms Y by Z%)
- Supplement written results with visual aids (e.g., charts or tables)

Discussion

- Analyzes your findings from a more qualitative perspective
- Answers one or more of the following questions:
  - What were some key themes across your findings?
  - What did your method get right?
  - What are some lingering areas for improvement?
  - Based on your findings, what recommendations would you make?
- Might include a more formal error analysis
  - What are some common error categories?
  - What percentage of instances belong to error category X?
  - What might have caused this category of errors?
  - How might it be addressed in the future?

Conclusions

- **Summarizes your findings and key takeaway points** — What do you want people to remember when they finish reading your paper?
- **Reiterates contributions**
- **Provides links to your data or code, if available**
- **Optionally suggests some high-level future directions**

Don’t neglect your references!

- Important to make sure your references are properly cited
  - Make sure all authors’ names are included
  - Make sure venue/journal names are accurate
  - Visually scan your reference section to make sure nothing looks “off”
- Check with your target venue to see how references should be formatted

How long should each section be?

- Depends on your paper contents and any length constraints set by your target venue
- As a general rule:
  - Introduction: 10%
  - Related Work: 15%
  - Methods: 35%
  - Evaluation: 25%
  - Discussion: 10%
  - Conclusion: 5%

Tools of the Trade

- Common text processors:
  - Microsoft Word
  - LaTeX
- LaTeX is the norm in most CS research fields
- If writing with a group (very common in NLP research!), check with your co-authors to see what their preferred workflow is for writing/editing:
  - In-text notes?
  - Tracked changes?
  - Comments?

LaTeX Basics

- Markup language that is compiled into a PDF
- Pros:
  - Produces professional-looking papers with minimal formatting intervention
  - Offers good support for typesetting equations
  - No need to worry about reference formatting!
- Cons:
  - Learning curve
  - Requires compiling to see what the final product looks like
- Popular platform for collaborative LaTeX editing: www.overleaf.com
- Check for venue-specific style files
  - ACM: https://www.acm.org/publications/proceedings-template
  - IEEE: https://www.ieee.org/conferences/publishing/templates.html
  - ACL: https://acl-org.github.io/ACLPUB/formatting.html
- Useful LaTeX documentation: https://www.overleaf.com/learn/latex/Learn_LaTeX_in_30_minutes

Tips for Collaborative Writing

Plan

Create a plan to distribute work among co-authors

- Popular option: Assign co-authors to sections based on which aspects of the research they worked on

Order

Decide on authorship order

- Convention varies depending on CS discipline
- Common in AI:
- For papers with joint first authors, indicate so with a footnote

Timeline

Discuss a timeline

- Decide when the first draft should be ready
- Agree on how much time each author needs to complete their tasks

Norms

Decide on norms for writing/editing

- Can take turns or edit all at once (if all at once, discuss how editing conflicts will be managed)

How to Edit a Research Paper

Start with easy edits:	• Run a spelling or grammar checker • Classic writing resource: Elements of Style
“Tighten” your writing:	• Are there places where you’re repeating yourself? • Are there simpler ways to describe things?
Scan for writing “flow”:	• Are there any unnatural or abrupt transitions? • Does the order of information make sense?
Don’t be afraid to delete large sections or rewrite heavily!	• A first draft is *not* a final draft (and usually is unrecognizable by the time you’re finished editing!)

What about the title?

Never underestimate the power of a good title!

Catchy titles may encourage people to skim through your paper or attend your conference presentation

Make sure your title isn't misleading or misrepresentative of the paper's contents

Try to keep your title concise while still remaining descriptive

Practice makes perfect!

Writing good research papers is an acquired skill---don’t worry if you’re not an expert yet!

Editing your peers’ papers can help you learn what to focus on and what to avoid

Volunteering to review for conferences and workshops in your research area can help you learn what reviewers are looking for from the “other side”

If your research area usually only recruits reviewers with advanced degrees, ask your advisor if you can serve as a secondary or “shadow” reviewer on one or two of their upcoming review assignments

Oral Presentations

Much like writing good research papers, giving good research presentations is the result of repeated practice

No one is born an excellent presenter!

Since NLP conferences tend to have lots of simultaneous paper presentations, the presentation length is often brief (~15 minutes)

This means that you must report your work clearly and concisely!

Planning Your Presentation

Think about the most important aspects of your work

- What absolutely needs to be communicated within this timeframe?

Think about your audience

- If presenting to experts, you may need to spend less time explaining the background/rationale; if explaining to a more novice audience, you may need to cut back on technical jargon

How to Create Good Presentation Slides

- Avoid including too much text on your slides
  - Slides should be used as visual aids to support what you're saying ...if they're too busy, people will end up paying more attention to the slides than to you!
- Figure out your average speaking duration per slide, and use that to determine how many slides you'll need overall (avoid rushing near the end!)
  - Common rule of thumb: one slide per minute
- Minimize the number of equations you include in your slides

Charts, Tables, and Graphs

- In most cases, you'll want to recreate tables from your paper using your slide editor
    - Screen captures can look sloppy, and tables included this way often include too much information
- Charts and graphs may be used directly if the font is legible and the formatting works well with the presentation
- Think about your presentation as a way to convince people to read your full paper

Talking Points

- Avoid reading directly from your slides
  - Not a good use of your audience’s time, and can be boring
- Avoid copying sentences directly from your paper
  - Concise bulletpoints work better for presentations
- Try to avoid speaking in a monotone (easier to do with practice!)
- Try to make eye contact with your audience
  - Usually there’ll be at least one “nodder” who you can find and talk to (when you’re attending conference talks, the presenter will be very grateful if you are that person!)

Presentation Jitters

Most people get nervous presenting---you are not alone!

Although you might feel like your voice is clearly shaky, most people who don't know you won't pick up on it

- People who do pick up on it will usually feel sympathetic anyway

Practicing your presentation ahead of time, even if just alone at home, can help cement the flow/content in your mind

- That way, if you're blanking out during the presentation, it's easier to go on autopilot

Q&A

Most conference presentations will include a short Q&A session after the talk

Try to start each answer by thanking the person asking the question (it makes them feel good!)

When answering a question, feel free to rephrase it a bit in a way that makes more sense to you

"Thanks for the insightful question! I think what you're getting at is…."

It's okay to admit that you don't know the answer to a question

You can always thank the person asking the question and let them know that you'll get back to them (much better than making something up!)

"Thank you for that question! It's something I haven't thought through yet, but very interesting to consider—I'd love to give it a bit more careful thought and reconnect with you later."

Other Presentation Formats You May Encounter

Poster Presentation

- Very common, especially at large NLP conferences
- Requires that you summarize your paper on a visually appealing poster, and prepare a "lightning talk" overview of the work
- Can allow for more direct 1:1 conversation about the work

Chalk Talk

- Not very common in NLP, but you may encounter it in other fields
- Requires that you dynamically explain your work using a chalkboard/dry erase board rather than slides

Panel

- Common in later stages of your career
- Typically requires that you prepare a brief presentation but spend most of the time answering questions, which you may or may not have access to ahead of time

Now you're well on your way to becoming an NLP researcher!

- Other paper-writing tips:
  - https://medium.com/@vered1986/tips-for-writing-nlp-papers-9c729a2f9e1f
- Some additional formal and informal mentoring opportunities that you may find helpful:
  - ACL Year-Round Mentorship: https://mentorship.aclweb.org/
  - ACL Student Research Workshop: https://2025.aclweb.org/calls/student_research_workshop/

Keeping up with NLP Research

NLP News by Sebastian Ruder https://www.ruder.io/nlp-news/

Follow NLP researchers and conference venues on social media

Mailing lists: https://aclweb.org/aclwiki/Newsgroups,_mailing_lists

Association for Computational Linguistics: https://www.aclweb.org/portal/

Summary: Prompt-Based Learning and Essentials of NLP Research

- 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
- One way to structure your systematic literature review is by loosely following the PRISMA guidelines
- When conducting research, make sure to form research questions that are specific, testable, and grounded in existing findings
- It is important to ensure that your research is reproducible, both for your own future benefit and that of others trying to extend your work
- The primary ways to communicate NLP research are through research papers and oral presentations, and both require repeated practice to master!