

LLM Agents

Slido: <https://app.sli.do/event/rzAMNVhnNM58rKG74nFaZd>

Yu Meng

University of Virginia
yumeng5@virginia.edu

Nov 19, 2025

Overview of Course Contents

- Week 1: Logistics & Overview
- Week 2: N-gram Language Models
- Week 3: Word Senses, Semantics & Classic Word Representations
- Week 4: Word Embeddings
- Week 5: Sequence Modeling & Recurrent Neural Networks (RNNs)
- Week 6: Language Modeling with Transformers
- Week 8: Transformer and Pretraining
- Week 9: Large Language Models (LLMs) & In-context Learning
- Week 10: Knowledge in LLMs and Retrieval-Augmented Generation (RAG)
- Week 11: LLM Reasoning
- Week 12: Reinforcement Learning for Post-Training LLMs
- **Week 13: LLM Alignment & Agents**
- Week 15 (after Thanksgiving): Project Presentations

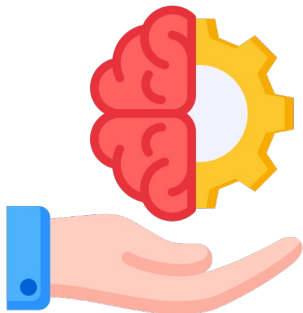
(Recap) The Evolution of GPT Models: ChatGPT

- GPT-1: decoder-only Transformer pretraining
- GPT-2: language model pretraining is multi-task learning
- GPT-3: scaling up & in-context learning
- ChatGPT: language model alignment



(Recap) Overview: Language Model Alignment

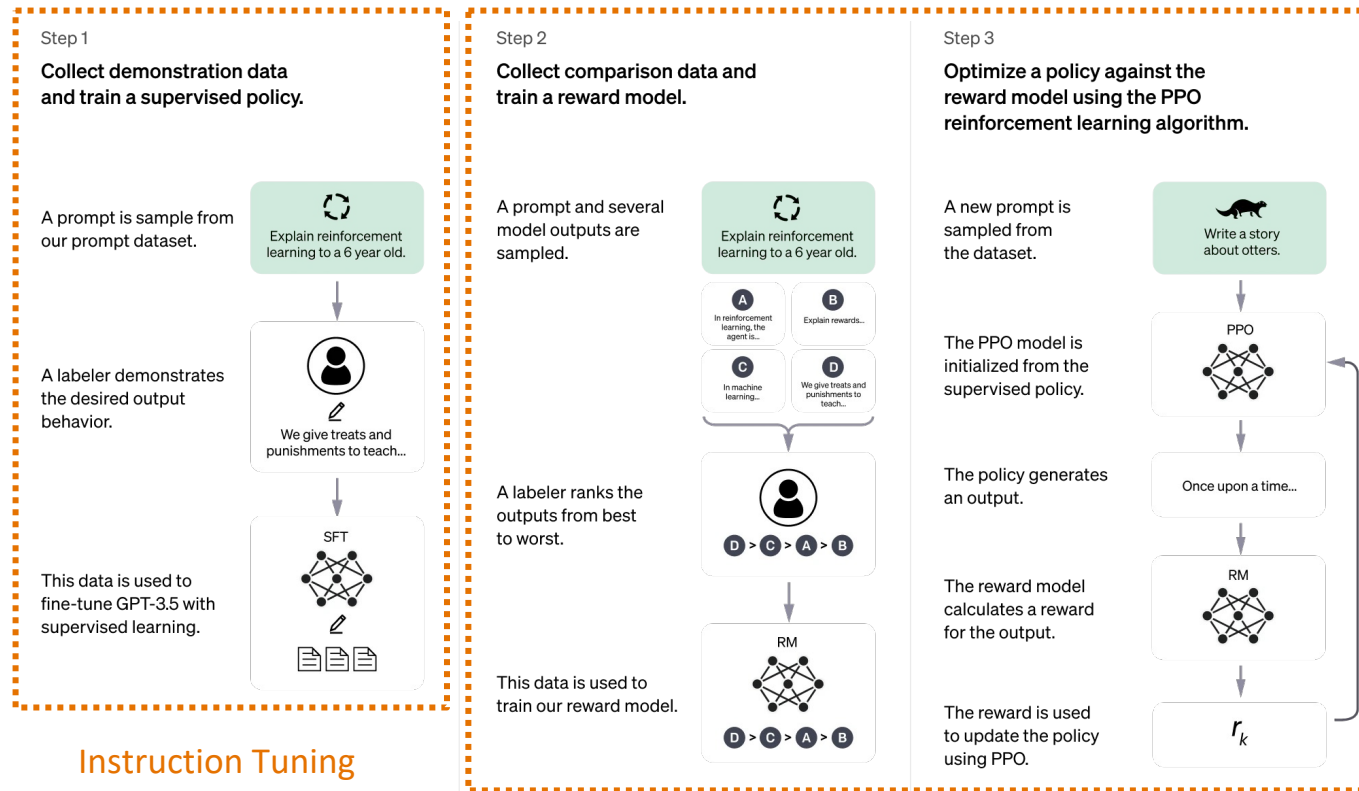
- Ensure language models behaviors are aligned with human values and intent for general tasks/applications
- “HHH” criteria (Askell et al. 2021):
 - **Helpful:** Efficiently perform the task requested by the user
 - **Honest:** Give accurate information & express uncertainty
 - **Harmless:** Avoid offensive/discriminatory/biased outputs



(Recap) Language Model Alignment: Post-training

- Pretrained language models are **not** aligned
- Objective mismatch
 - Pretraining is to predict the next word in a sentence
 - Does not involve understanding human intent/values
- Training data bias
 - Text from the internet can contain biased, harmful, or misleading information
 - LMs don't distinguish between good and bad behavior in training data
- (Over-)generalization issues
 - LMs' generalization can lead to outputs that are inappropriate in specific contexts
 - Might not align with intended ethics/honesty standard

(Recap) Language Model Alignment Techniques



Reinforcement
Learning from
Human Feedback
(RLHF)

(Recap) Overview: Instruction Tuning

- Train an LM using a diverse set of tasks
 - Each task is framed as an **instruction** followed by an example of the desired output
 - The goal is to teach the model to follow specific instructions (human intent) effectively
- The resulting model can perform a variety of tasks **zero-shot** (w/o requiring in-context demonstrations)
- The instructions can also be in chat format – tuning an LM into a chatbot

🔗 meta-llama/Llama-3.2-1B

📄 Text Generation • Updated 8 days ago • 📄 1.05M • ⚡ • ❤️ 725

Pretrained (base) model

🔗 meta-llama/Llama-3.2-1B-Instruct

📄 Text Generation • Updated 8 days ago • 📄 1.31M • ⚡ • ❤️ 478

Instruction-tuned
(post-trained) model



(Recap) Instruction Tuning: Introduction

- **Setting:** fine-tune LLMs with task-specific instructions on diverse tasks
- **Goal:** enable LLM to better understand user prompts and generalize to a wide range of (unseen) tasks **zero-shot**

FINETUNED LANGUAGE MODELS ARE ZERO-SHOT LEARNERS

**Jason Wei*, Maarten Bosma*, Vincent Y. Zhao*, Kelvin Guu*, Adams Wei Yu,
Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le**
Google Research

(Recap) Instruction Tuning: Method

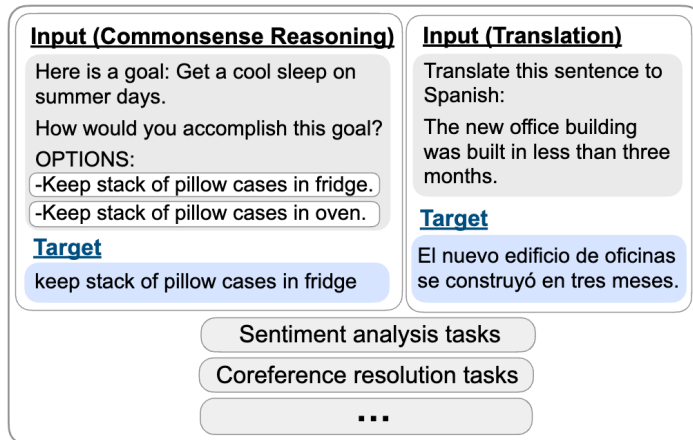
- **Input:** task description
- **Output:** expected response or solution to the task
- Train LLMs to generate response tokens given prompts

$$\min_{\theta} -\log p_{\theta}(y|x)$$

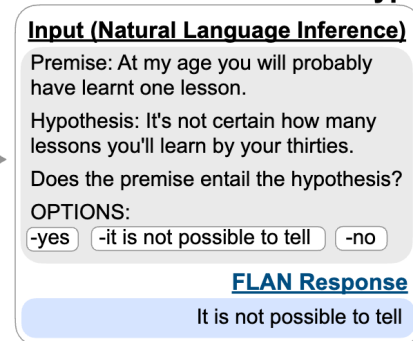
Response

Prompt

Finetune on many tasks (“instruction-tuning”)



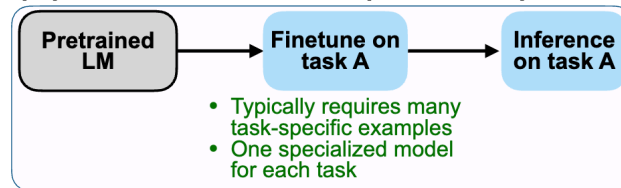
Inference on unseen task type



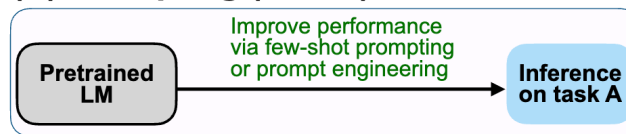
(Recap) Instruction Tuning vs. Other Paradigms

- Task-specific fine-tuning does not enable generalization across multiple tasks
- In-context learning requires few-shot demonstrations
- Instruction tuning enables zero-shot cross task generalization

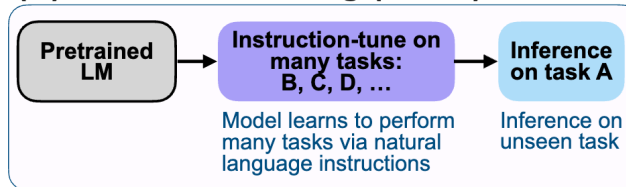
(A) Pretrain–finetune (BERT, T5)



(B) Prompting (GPT-3)



(C) Instruction tuning (FLAN)

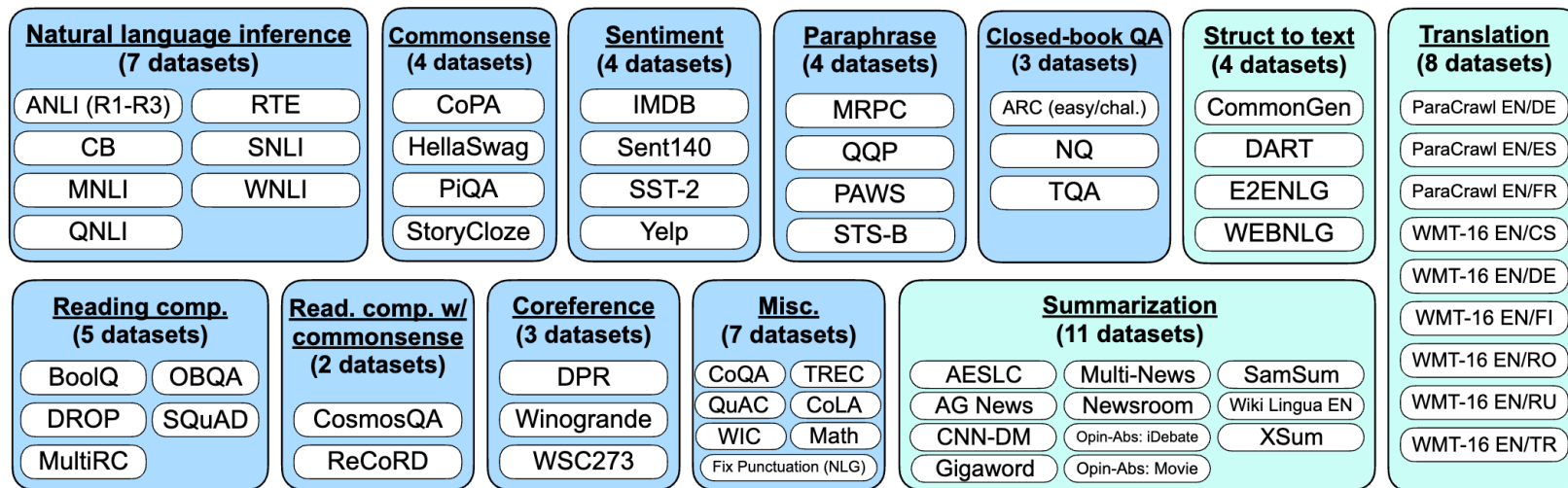


(Recap) Instruction Tuning vs. Pretraining

- Both instruction tuning and pretraining are **multi-task** learning paradigms
- Supervision
 - Pretraining: self-supervised learning (raw data w/o human annotation)
 - Instruction tuning: supervised learning (human annotated responses)
- Task format
 - Pretraining: tasks are implicit (predicting next tokens)
 - Instruction tuning: tasks are explicit (defined using natural language instructions)
- Goal
 - Pretraining: teach LMs a wide range of linguistic patterns & general knowledge
 - Instruction tuning: teach LMs to follow specific instructions and perform a variety of tasks

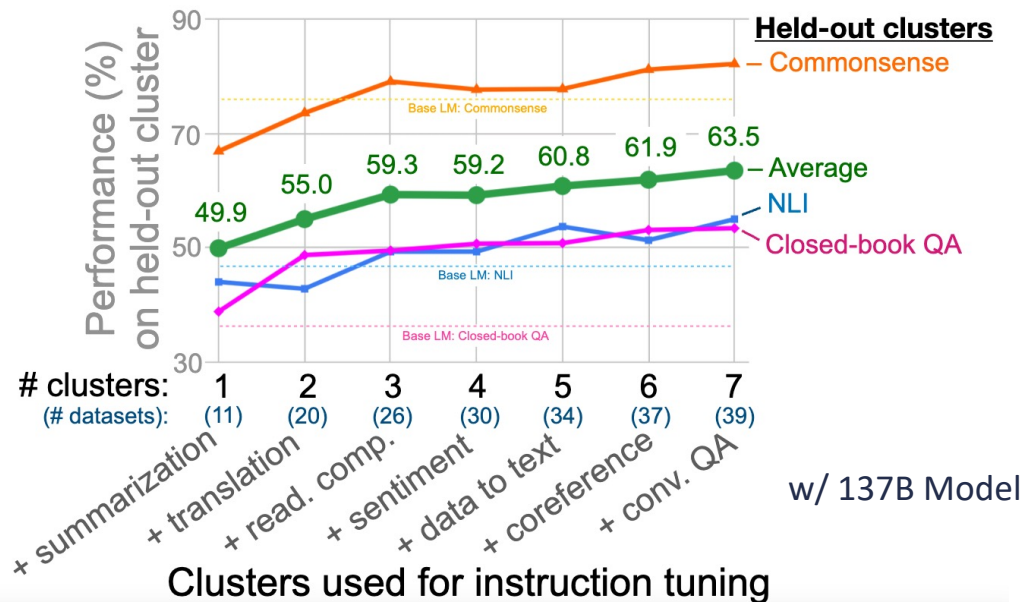
(Recap) FLAN: Collection of Instruction Tuning Datasets

62 datasets (12 task clusters) covering a wide range of understanding + generation tasks



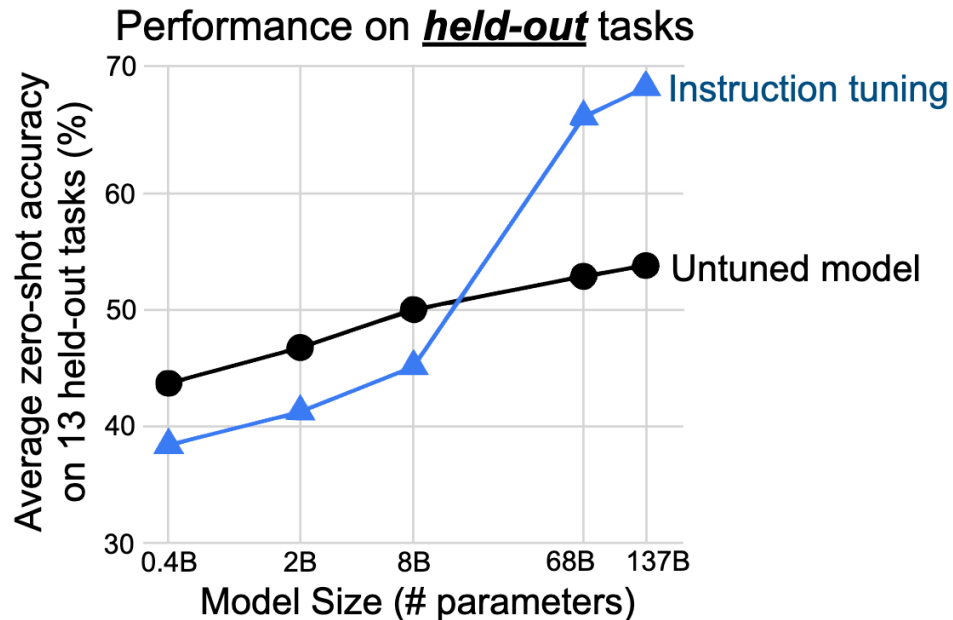
(Recap) Generalization Improves with More Clusters

- Held out three clusters from instruction tuning: Commonsense, NLI, Closed-book QA
- More clusters and tasks used in instruction tuning => better generalization to unseen clusters



(Recap) Instruction Tuning with Different Model Sizes

- Instruction tuning can hurt small model ($< 8\text{B}$) generalization
- Instruction tuning substantially improves generalization for large models



(Recap) Chat-style Instruction Tuning

- Instruction tuning can also be used to build chatbots for multi-turn dialogue
- Instructions may not correspond strictly to one NLP task, but mimic a human-like dialogue
- Multi-turn instruction tuning training data example:

```
{"role": "user", "content": "What's the weather like today?"},  
{"role": "assistant", "content": "It's sunny with a high of 75 degrees."},  
{"role": "user", "content": "Great! What about tomorrow?"},  
{"role": "assistant", "content": "Tomorrow will be partly cloudy with a high of 72 degrees."}
```

(Recap) Limitations of Instruction Tuning & Why RLHF

- **Costly human annotations**
 - Instruction tuning requires human annotators to write down the entire expected responses
 - RLHF only relies on preference labels (which response is better?)
- **Open-ended generation**
 - Open-ended creative generation (e.g., story writing) inherently has no single “right” answer
 - RLHF uses human feedback to determine which response is more creative/appealing
- **Token-level learning**
 - Instruction tuning applies the language modeling loss -> penalizes all token mistakes equally regardless of their impact on the overall quality of the output (e.g., a grammatical error might be less critical than a factual inaccuracy)
 - RLHF uses human feedback to prioritize the error types that are more important to correct
- **Suboptimal human answers**
 - Instruction tuning may learn the suboptimal patterns written by humans
 - Identifying a better answer from a few options is usually easier than writing an optimal answer entirely



(Recap) Overview: RLHF

- Human feedback collection
 - Generate multiple responses using the model given the same prompt
 - Human evaluators rank responses of the model based on helpfulness/honesty/safety...
- Reward model training
 - A reward model is trained on human feedback data to predict the quality of responses
 - Higher reward = more preferred by human evaluators
- Policy optimization
 - Use reinforcement learning algorithms to further train the LM to maximize the reward predicted by the reward model
 - Encourage the model to produce outputs that align better with human preferences

**Training language models to follow instructions
with human feedback**

(Recap) RLHF Illustration

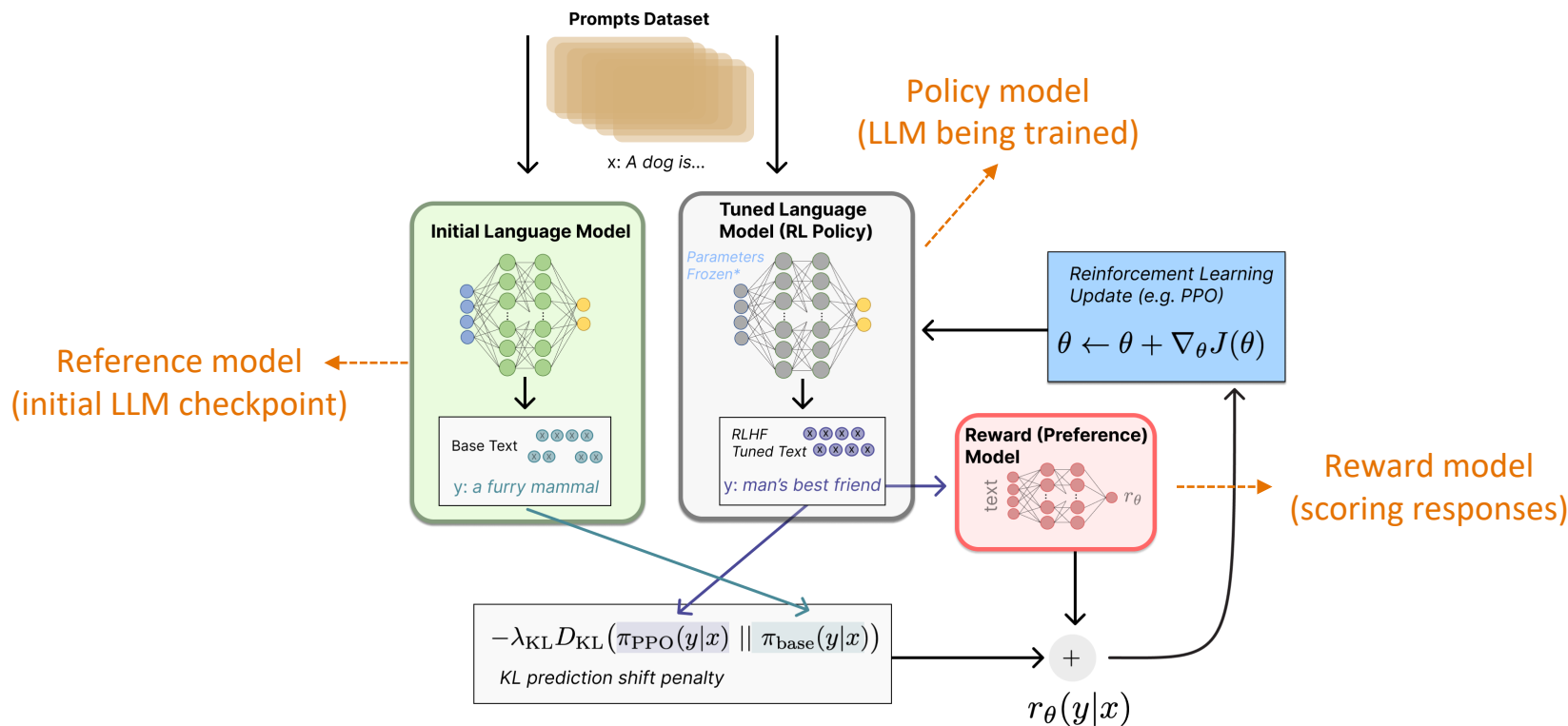


Figure source: <https://huggingface.co/blog/rlhf>

(Recap) Preference Data Construction

- Goal of reward model: score the quality of LLM's output based on human feedback
- Can we directly ask human annotators to assign a scalar score (e.g., 1-10) to a single response?

What are the steps for making a simple cake?

1. *Warm up the oven.*
2. *Grease a cake pan.*
3. *Blend dry ingredients in a bowl.*
4. *Incorporate butter, milk, and vanilla.*
5. *Mix in the eggs.*
6. *Pour into the prepared pan.*
7. *Bake until golden brown.*
8. *Add frosting if desired.*

Different human evaluators
can be very inconsistent in
assigning absolute scores!

(Recap) Preference Data with Pairwise Comparisons

Humans are better at relative judgments than absolute ones

What are the steps for making a simple cake?

1. Preheat oven to 350°F (175°C).
2. Grease and flour a cake pan.
3. In a bowl, combine 2 cups flour, 1.5 cups sugar, 3.5 tsp baking powder, and a pinch of salt.
4. Add 1/2 cup butter, 1 cup milk, and 2 tsp vanilla; mix well.
5. Beat in 3 eggs, one at a time.
6. Pour batter into the pan.
7. Bake for 30-35 minutes or until a toothpick comes out clean.
8. Let cool, then frost or serve as desired.

What are the steps for making a simple cake?

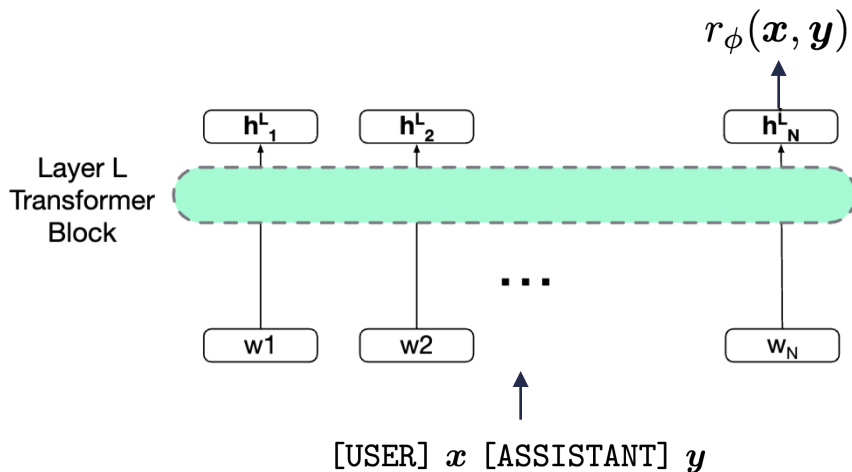
1. Warm up the oven.
2. Grease a cake pan.
3. Blend dry ingredients in a bowl.
4. Incorporate butter, milk, and vanilla.
5. Mix in the eggs.
6. Pour into the prepared pan.
7. Bake until golden brown.
8. Add frosting if desired.

Preference data: (x, y_w, y_l)



Reward Model Setup

Goal: train a reward model to assign a higher reward to y_w than y_l



Apply a linear layer at the last token representation to learn a scalar output



Reward Model Training

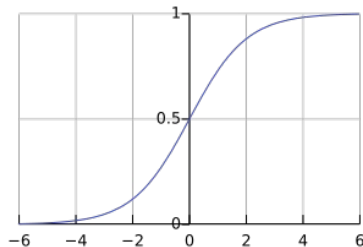
Bradley-Terry pairwise comparison objective

$$\mathcal{L}_{\text{RM}}(r_\phi) = -\mathbb{E}_{(\mathbf{x}, \mathbf{y}_w, \mathbf{y}_l) \sim \mathcal{D}} [\log \sigma(r_\phi(\mathbf{x}, \mathbf{y}_w) - r_\phi(\mathbf{x}, \mathbf{y}_l))]$$

reward of winning
response

reward of losing
response

$$y = \sigma(x)$$



Optimizing LLMs with the Reward Model

- The trained reward model serve as a proxy for human judgment (higher reward = more preferred by humans)
- Maximize the reward of generated responses from the LLM (policy model)

$$\max_{\theta} \mathbb{E}_{\underset{\text{LLM output}}{\underset{\text{probability}}{\mathbf{y} \sim p_{\theta}(\cdot | \mathbf{x})}}} [\underset{\text{reward of LLM}}{\underset{\text{generated response}}{r_{\phi}(\mathbf{x}, \mathbf{y})}]$$

- What if our reward model is imperfect?

Issues with Naïve Optimization of Rewards

- Reward models are still only **approximations** of true human preferences
 - Can be noisy or incomplete (e.g., not well-generalized out-of-domain)
- Solely maximizing the reward leads to several issues
 - **Exploiting reward model flaws:** The LLM might learn to “hack” the reward model, finding ways to achieve high reward without actually possessing the desired behavior
 - **Mode collapse:** The LLM might converge to a narrow distribution of outputs that achieve high reward, but lack diversity and fail to generalize to different situations
 - **Loss of pretrained knowledge:** Over-optimization for the reward model can cause the LLM to unlearn desirable properties in the initial pretrained model (e.g., grammar, factuality)

Regularized Reward Optimization

- Add a penalty for drifting too far from the initial SFT checkpoint

$$\max_{\theta} \mathbb{E}_{\mathbf{y} \sim p_{\theta}(\cdot|\mathbf{x})} \left[\underbrace{r_{\phi}(\mathbf{x}, \mathbf{y})}_{\text{Maximize reward}} - \underbrace{\beta \log \left(\frac{p_{\theta}(\mathbf{y}|\mathbf{x})}{p_{\text{SFT}}(\mathbf{y}|\mathbf{x})} \right)}_{\text{Prevent deviation from the initial (SFT) model}} \right]$$

hyperparameter

- Penalize cases where $p_{\theta}(\mathbf{y}|\mathbf{x}) > p_{\text{SFT}}(\mathbf{y}|\mathbf{x})$
- In expectation, it is known as the Kullback-Leibler (KL) divergence $\text{KL}(p_{\theta}(\mathbf{y}|\mathbf{x}) || p_{\text{SFT}}(\mathbf{y}|\mathbf{x}))$

Optimization with Reinforcement Learning (RL)

- Why reinforcement learning:
 - No supervised data available (only a reward model)
 - Encourage the model to explore new possibilities (generations) guided by the reward model
- Optimization: policy gradient methods
 - Optimize the policy (LLM) by adjusting the parameters in the direction that increases expected rewards
- REINFORCE (simplest policy gradient method):

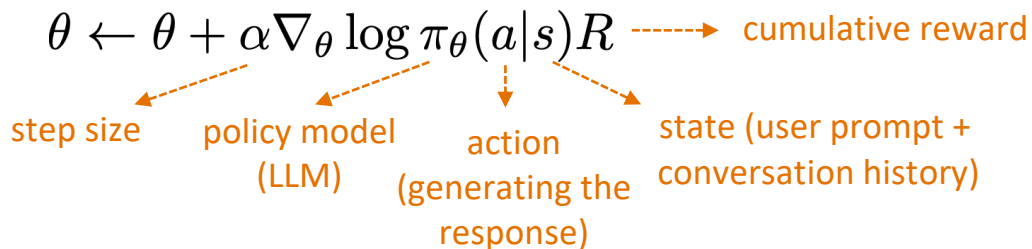
$$\theta \leftarrow \theta + \alpha \nabla_{\theta} \log \pi_{\theta}(a|s) R$$


Diagram illustrating the REINFORCE equation components:

- α : step size
- $\pi_{\theta}(a|s)$: policy model (LLM)
- a : action (generating the response)
- s : state (user prompt + conversation history)
- R : cumulative reward

Overview: Direct Preference Optimization (DPO)

- Overall, the RLHF framework is very complicated
 - Need to first train a reward model
 - Need to do online sampling
 - Performance is very sensitive to many hyperparameters
- Direct Preference Optimization (DPO): optimize LM parameters directly on preference data by solving a binary classification problem (without an explicit reward model)

Direct Preference Optimization: Your Language Model is Secretly a Reward Model

Rafael Rafailov^{*†}

Archit Sharma^{*†}

Eric Mitchell^{*†}

Stefano Ermon^{†‡}

Christopher D. Manning[†]

Chelsea Finn[†]

[†]Stanford University [‡]CZ Biohub
{rafailov,architsh,eric.mitchell}@cs.stanford.edu

Paper: <https://arxiv.org/pdf/2305.18290>

Further Reading on RLHF

- [RAFT: Reward rAnked FineTuning for Generative Foundation Model Alignment](#) [Dong et al., 2023]
- [Iterative Preference Learning from Human Feedback: Bridging Theory and Practice for RLHF under KL-Constraint](#) [Xiong et al., 2023]
- [SLiC-HF: Sequence Likelihood Calibration with Human Feedback](#) [Zhao et al., 2023]
- [SimPO: Simple Preference Optimization with a Reference-Free Reward](#) [Meng et al., 2024]

Agenda

- LLM Agent Overview
- Multimodal LLMs
- Tool Usages
- Code Assistant

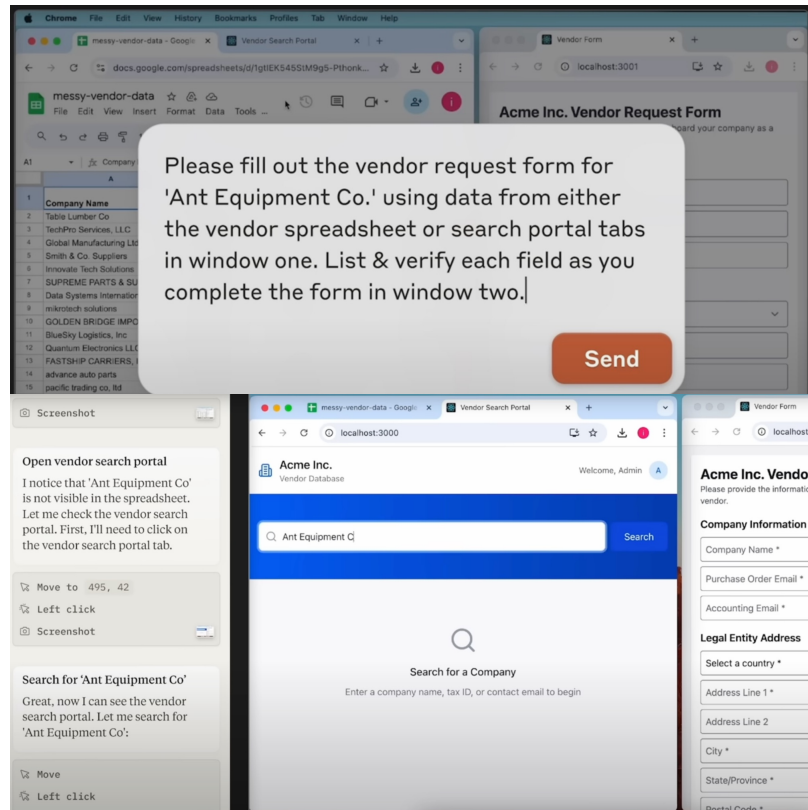
Overview: Language Agents

- Language agents: systems that interact with users using natural language as an interface to execute real-world tasks
- LLMs serve as the foundation for language agents
 - **Natural language understanding:** comprehend and interpret user input in text
 - **Natural language generation:** generate coherent & appropriate responses/actions
 - **Reasoning:** enable multi-step reasoning or problem-solving/decision-making
- Examples:
 - **Virtual assistants:** understand user commands and carry out tasks (e.g., setting reminders, playing music, controlling smart home devices)
 - **Code agents:** assist developers by generating code snippets, suggesting improvements, and explaining how certain pieces of code work
 - **Business operations:** break down high-level goals (e.g., “create a marketing campaign”), search and synthesize information, and execute steps autonomously (e.g., interacting with external API/tools)

Claude 3.5: Computer Use

Introducing computer use, a new Claude 3.5 Sonnet, and Claude 3.5 Haiku

Oct 22, 2024 • 5 min read



WebShop: Language Agents for Online Shopping

Instruction: i am looking for x-large, red color women faux fur lined winter warm jacket coat, and price lower than 70.00 dollars

Current Query: women fur jacket coat

Results

Page 1 (1-10) of 50 total results

[Back to Search](#)

[Next >](#)



B09KP78G37

Women Faux Fur Lined Jacket Coat
Winter Warm Thick Fleece Outwear
Trench Zipper Plus Size Long
Sleeve Blush Overcoat



B07ZXBGDXF

Women's Coat, FORUU Winter Faux
Fur Fleece Outwear Warm Lapel
Biker Motor Aviator Jacket



B098XT346Y

Fjackets Real Lambskin Sherpa
Jacket - Mens Leather Jacket

★★★★★ 4.7

Current Action: click[Back to Search]

Agenda

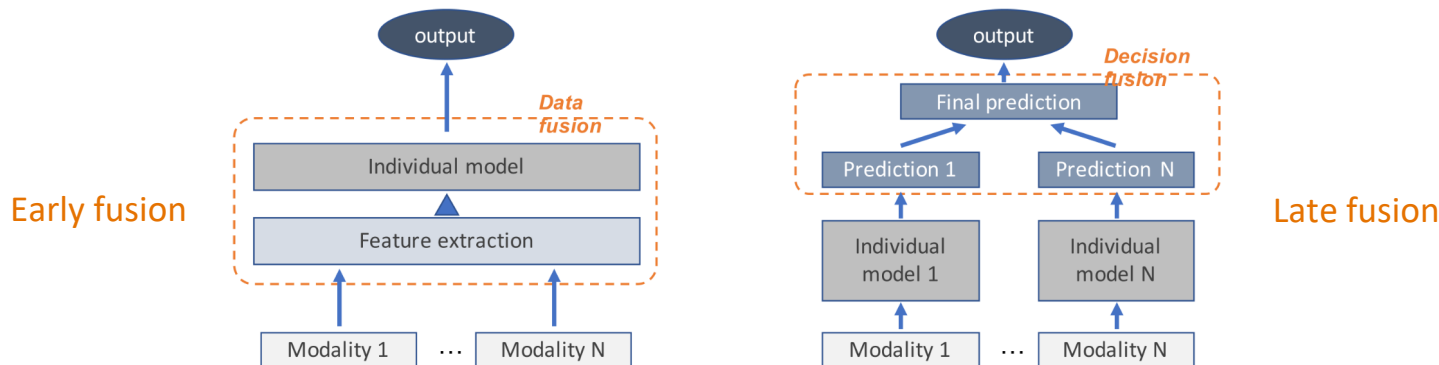
- LLM Agent Overview
- Multimodal LLMs
- Tool Usages
- Code Assistant

Overview: Multimodal LLMs

- Process and understand multiple types of data (e.g., text, images, audio, and video)
- More comprehensive and contextually rich understanding & generation
- Multimodal input processing (common):
 - Accept and process different types of input data
 - Examples: understanding the content of an image, transcribing and interpreting speech, analyzing video content, or integrating information from sensor data
- Multimodal output generation (less common):
 - Generate output in various modalities
 - Examples: creating realistic images from text descriptions, translating speech to text, or generating music according to user descriptions

Overview: Multimodal Architecture

- Architecture:
 - Require modality-specific architectures (e.g., vision/audio/video encoders)
 - Usually LLMs serve as the strong base
- Multimodal fusion: fuse information from different modalities
 - Early fusion: Combine raw input data from different modalities before processing
 - Late fusion: Process each modality separately and then combine the representations later





Overview: Multimodal Datasets

Training datasets need to contain paired examples of different modalities => teach the model the relationships between different types of data



Source: <https://www.barnorama.com/wp-content/uploads/2016/12/03-Confusing-Pictures.jpg>

What is unusual about this image?

The unusual aspect of this image is a man ironing clothes on the back of a minivan or van. This is not a typical place to perform this activity, as one would usually iron clothes in a more stationary and safe location, such as a home, using a regular ironing board.

Figure source: <https://arxiv.org/pdf/2304.08485>



Visual Instruction Tuning

- Goal: fine-tune a multimodal LLM to learn to follow instructions for tasks that involve both visual and textual information
- **LLaVA** (Large Language and Vision Assistant): combine a pretrained vision encoder (e.g., CLIP) with a large language model (e.g., Llama) for visual instruction tuning

Visual Instruction Tuning

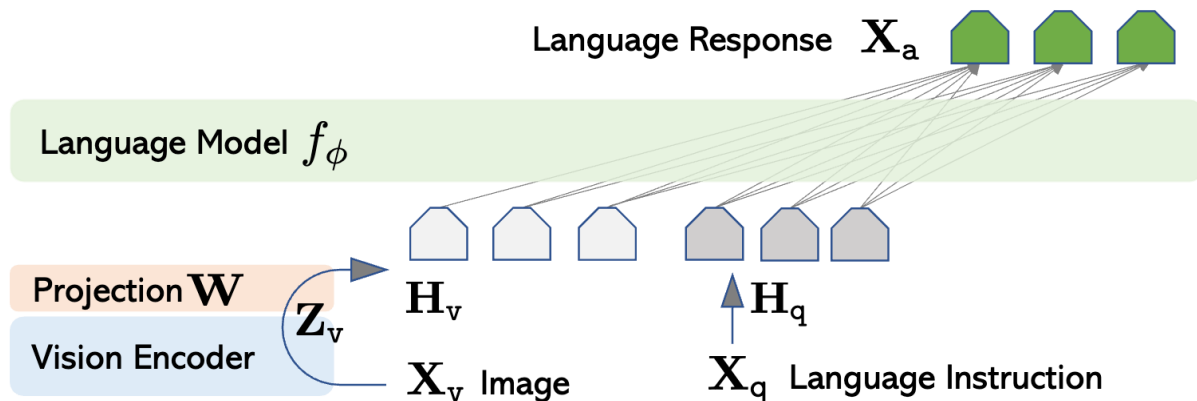
Haotian Liu^{1*}, Chunyuan Li^{2*}, Qingyang Wu³, Yong Jae Lee¹

¹University of Wisconsin–Madison ²Microsoft Research ³Columbia University

<https://llava-vl.github.io>

LLaVA: Architecture

- Learn a projection matrix ($\mathbf{W}$) to convert image representations ($\mathbf{Z}_v$) to text embeddings ($\mathbf{H}_v$)
- Concatenate visual tokens ($\mathbf{H}_v$) with text tokens ($\mathbf{H}_q$) as input to the model



Adopted in latest
multimodal Llama models

[meta-llama/Llama-3.2-90B-Vision](#)

[meta-llama/Llama-3.2-11B-Vision](#)

LLaVA: Results

Sometimes I just look at pictures of the earth from space and I marvel at how beautiful it all is.



User
LLaVA

Can you explain this meme in detail?

The meme in the image is a creative and humorous take on food, with a focus on chicken nuggets, as the center of the universe. The meme begins with the phrase "Sometimes I just look at pictures of the Earth from space and I marvel at how beautiful it all is..." The main image features a tray of cooked chicken nuggets, with the chicken pieces positioned to resemble the continents and islands on a world map. The meme playfully suggests that the chicken nuggets represent the Earth, and the various locations depicted in the photo are

Agenda

- LLM Agent Overview
- Multimodal LLMs
- Tool Usages
- Code Assistant



Tool Usages with LLMs

- Motivation: many task execution requires accessing & using external tools (e.g., calculator, calendar, search engines)
- **Toolformer**: train LMs to use various tools and automatically decide when and how to use which tool

Toolformer: Language Models Can Teach Themselves to Use Tools

**Timo Schick Jane Dwivedi-Yu Roberto Dessì[†] Roberta Raileanu
Maria Lomeli Luke Zettlemoyer Nicola Cancedda Thomas Scialom**

Meta AI Research [†]Universitat Pompeu Fabra

Types of Tools Considered

Automatically decide when & which tool to use during text generation

Question answering system

The New England Journal of Medicine is a registered trademark of [QA("Who is the publisher of The New England Journal of Medicine?") → Massachusetts Medical Society] the MMS.

Calculator

Out of 1400 participants, 400 (or [Calculator(400 / 1400) → 0.29] 29%) passed the test.

Machine translation

The name derives from "la tortuga", the Spanish word for [MT("tortuga") → turtle] turtle.

Wikipedia search

The Brown Act is California's law [WikiSearch("Brown Act") → The Ralph M. Brown Act is an act of the California State Legislature that guarantees the public's right to attend and participate in meetings of local legislative bodies.] that requires legislative bodies, like city councils, to hold their meetings open to the public.

Tool Learning via In-context Learning

- Provide example API calls in context
- LLMs learn to generate API calls for new data

In-context examples

Your task is to add calls to a Question Answering API to a piece of text. The questions should help you get information required to complete the text. You can call the API by writing "[QA(question)]" where "question" is the question you want to ask. Here are some examples of API calls:

Input: Joe Biden was born in Scranton, Pennsylvania.

Output: Joe Biden was born in [QA("Where was Joe Biden born?")] Scranton, [QA("In which state is Scranton?")] Pennsylvania.

Input: Coca-Cola, or Coke, is a carbonated soft drink manufactured by the Coca-Cola Company.

Output: Coca-Cola, or [QA("What other name is Coca-Cola known by?")] Coke, is a carbonated soft drink manufactured by [QA("Who manufactures Coca-Cola?")] the Coca-Cola Company.

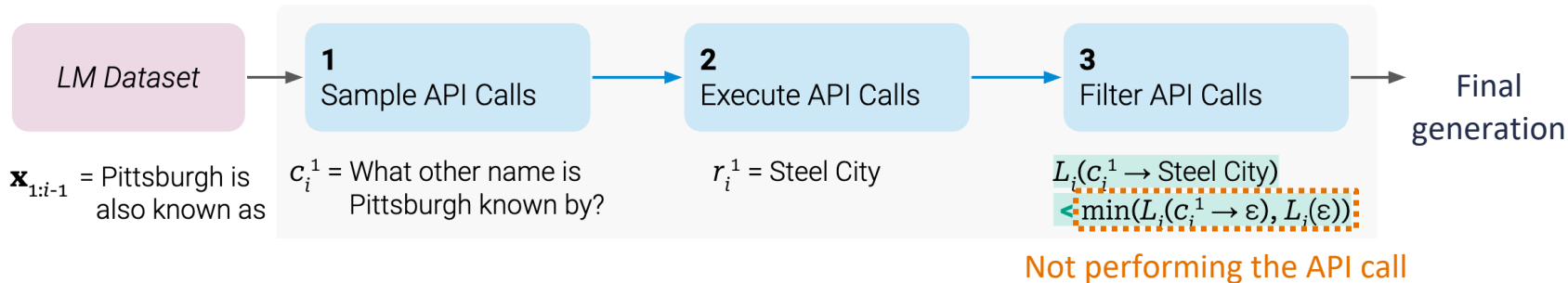
Input: x

Output:

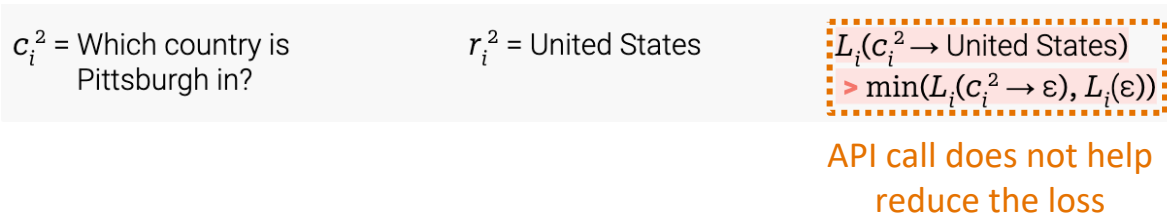
Generate API calls for new data

Filtering API Calls

- Some API calls are beneficial for the LLM to execute the task, while others are not
- Helpful API calls typically reduce the loss for generating future tokens

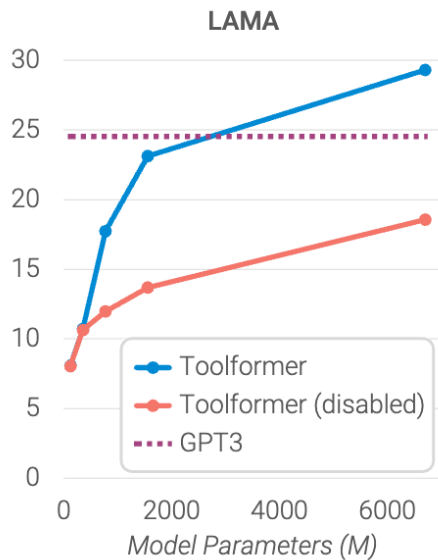


- Filter out API calls which do not reduce the loss

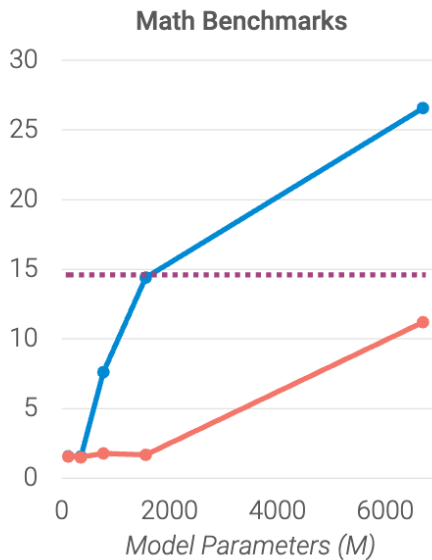


Tool Usage Ability vs. Model Scale

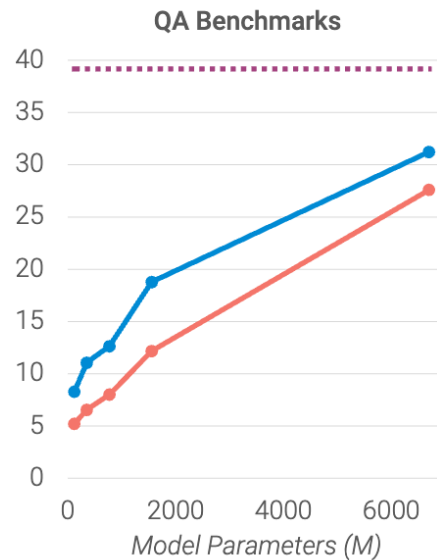
Larger models more effectively learn how to appropriately use tools



Wikipedia search



Calculator

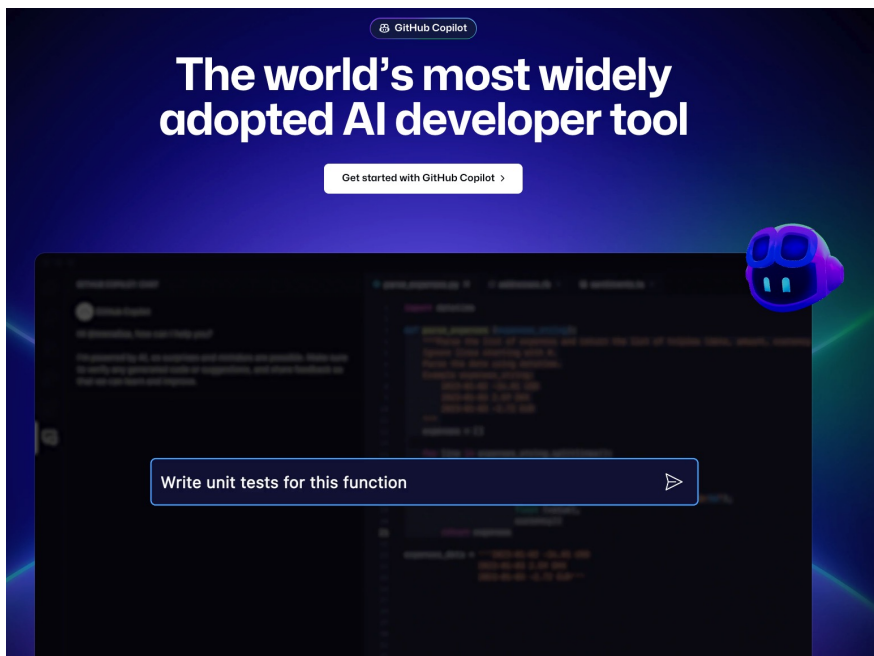


Question answering

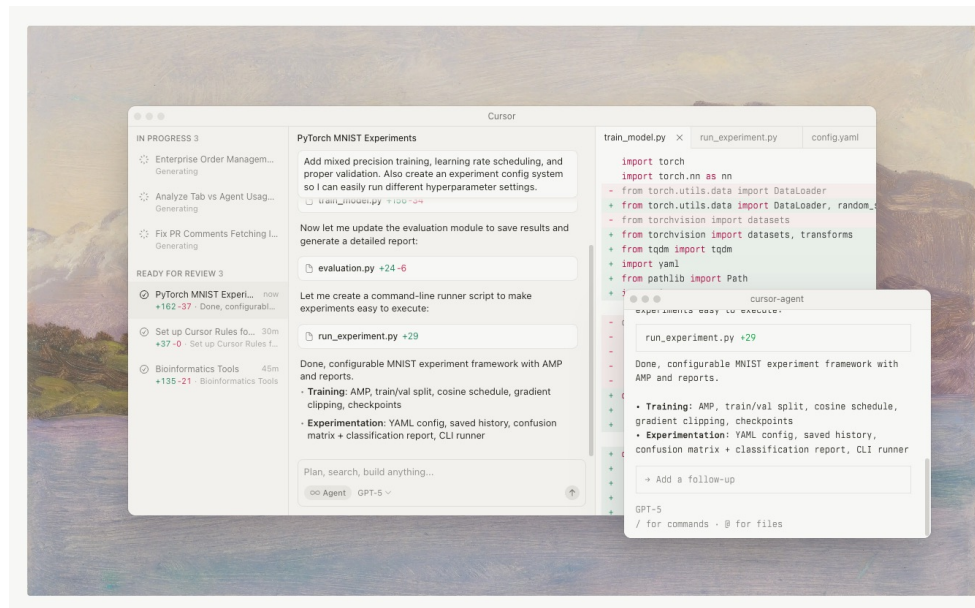
Agenda

- LLM Agent Overview
- Multimodal LLMs
- Tool Usages
- Code Assistant

LLMs as Code Assistants



<https://github.com/features/copilot>



<https://www.cursor.com/>

Code Infilling

- Motivation: code is seldom written in a single left-to-right pass and is instead repeatedly edited and refined
- Need train an LLM to perform both left-to-right code generation and editing (masking and infilling)

INCODER: A GENERATIVE MODEL FOR CODE INFILLING AND SYNTHESIS

Daniel Fried^{*♡†◇} Armen Aghajanyan^{*♡} Jessy Lin⁺
Sida Wang[♡] Eric Wallace⁺ Freda Shi[△] Ruiqi Zhong⁺
Wen-tau Yih[♡] Luke Zettlemoyer^{♡†} Mike Lewis[♡]
Facebook AI Research[♡] University of Washington[†]
UC Berkeley⁺ TTI-Chicago[△] Carnegie Mellon University[◇]
dfried@cs.cmu.edu, {armenag,mikelewis}@fb.com

InCoder Training: Causal Masking

- Sample several spans of code in training documents
- Move these spans to the end of the document, with their original location denoted by special mask tokens
- LLM is trained to produce these entire masked documents => learn to generate insertion text conditioned on bidirectional context

Original Document

```
def count_words(filename: str) -> Dict[str, int]:
    """Count the number of occurrences of each word in the file."""
    with open(filename, 'r') as f:
        word_counts = {}
        for line in f:
            for word in line.split():
                if word in word_counts:
                    word_counts[word] += 1
                else:
                    word_counts[word] = 1
    return word_counts
```

Masked Document

```
def count_words(filename: str) -> Dict[str, int]:
    """Count the number of occurrences of each word in the file."""
    with open(filename, 'r') as f:
        <MASK:0> in word_counts:
            word_counts[word] += 1
        else:
            word_counts[word] = 1
    return word_counts
<MASK:0> word_counts = {}
    for line in f:
        for word in line.split():
            if word <EOM>
```

InCoder Inference: Code Editing

Various types of code editing: insert mask tokens at desired locations and use the model to generate content to be inserted

Type Inference

```
def count_words(filename: str) -> Dict[str, int]:
    """Count the number of occurrences of each word in the file."""
    with open(filename, 'r') as f:
        word_counts = {}
        for line in f:
            for word in line.split():
                if word in word_counts:
                    word_counts[word] += 1
                else:
                    word_counts[word] = 1
    return word_counts
```

Variable Name Prediction

```
def count_words(filename: str) -> Dict[str, int]:
    """Count the number of occurrences of each word in the file."""
    with open(filename, 'r') as f:
        word_count = {}
        for line in f:
            for word in line.split():
                if word in word_count:
                    word_count[word] += 1
                else:
                    word_count[word] = 1
    return word_count
```

Docstring Generation

```
def count_words(filename: str) -> Dict[str, int]:
    """
    Counts the number of occurrences of each word in the given file.

    :param filename: The name of the file to count.
    :return: A dictionary mapping words to the number of occurrences.
    """
    with open(filename, 'r') as f:
        word_counts = {}
        for line in f:
            for word in line.split():
                if word in word_counts:
                    word_counts[word] += 1
                else:
                    word_counts[word] = 1
    return word_counts
```

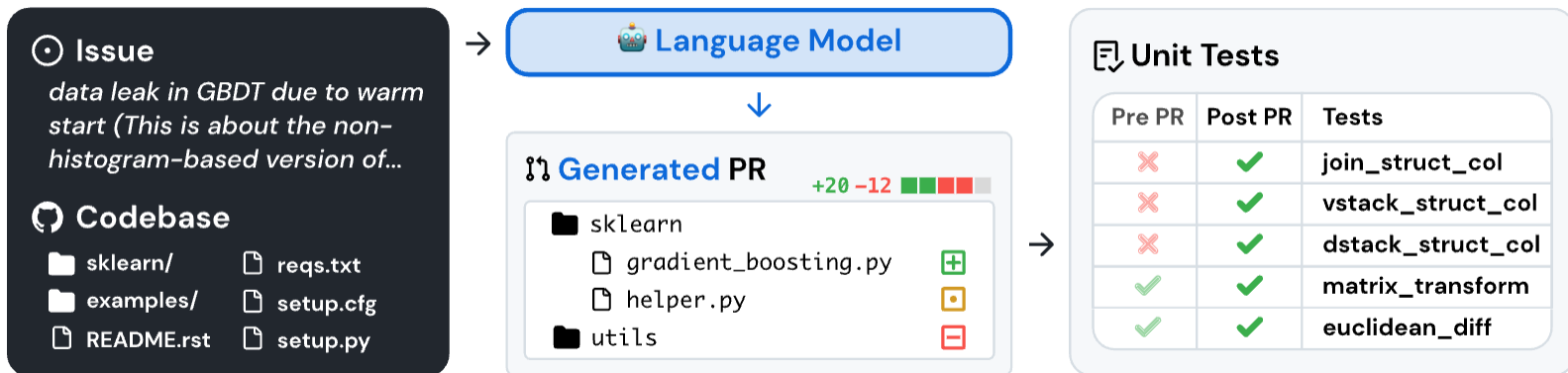
Multi-Region Infilling

```
from collections import Counter

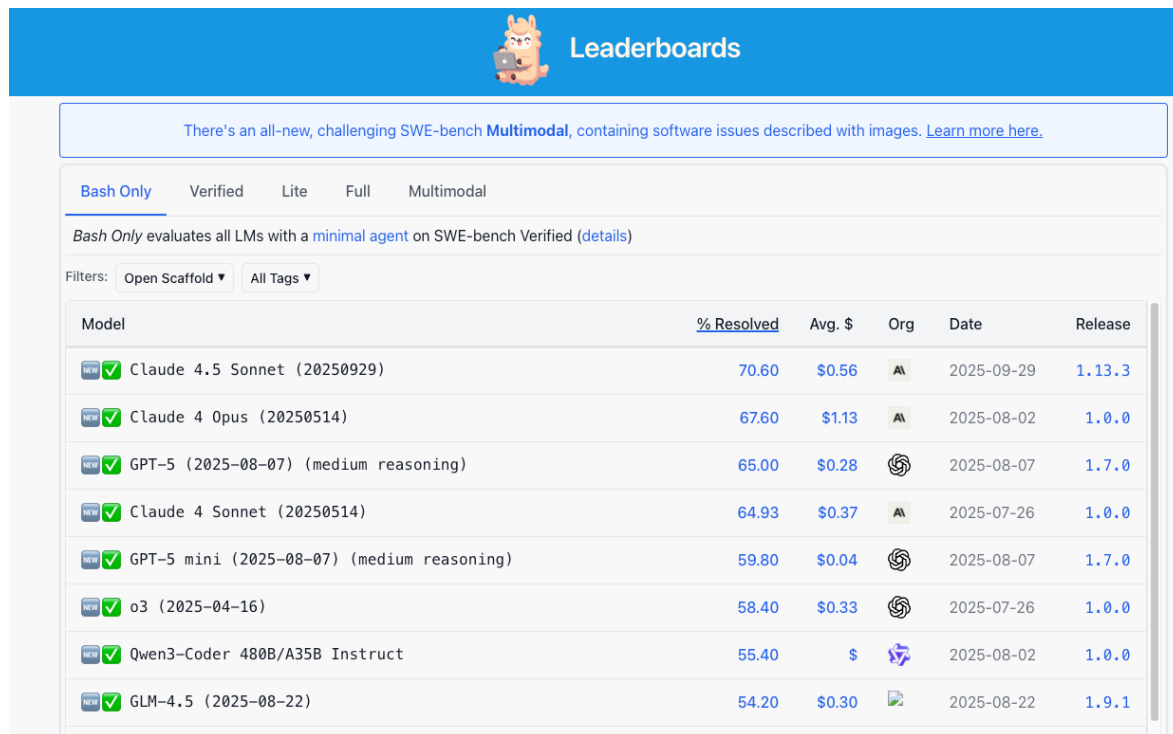
def word_count(file_name):
    """Count the number of occurrences of each word in the file."""
    words = []
    with open(file_name) as file:
        for line in file:
            words.append(line.strip())
    return Counter(words)
```

Code Agent Evaluation: SWE-Bench

- Collect task instances from real-world Python repositories by connecting GitHub issues to merged pull request solutions that resolve related test
- Provided with the issue text and a codebase snapshot, LLMs generate a patch that is evaluated against real tests



Latest Progress (As of 11/19/25) on SWE-Bench



The image shows the SWE-Bench Leaderboards page. At the top, there's a blue header with a cartoon dog icon and the word "Leaderboards". Below this, a light blue box contains a message about a new "Multimodal" benchmark. The main content area has tabs for "Bash Only", "Verified", "Lite", "Full", and "Multimodal". The "Bash Only" tab is selected, and a note states it evaluates all LMs with a "minimal agent". Below the tabs are filters for "Open Scaffold" and "All Tags". The table below lists various models with their performance metrics.

Model	% Resolved	Avg. \$	Org	Date	Release
  Claude 4.5 Sonnet (20250929)	70.60	\$0.56		2025-09-29	1.13.3
  Claude 4 Opus (20250514)	67.60	\$1.13		2025-08-02	1.0.0
  GPT-5 (2025-08-07) (medium reasoning)	65.00	\$0.28		2025-08-07	1.7.0
  Claude 4 Sonnet (20250514)	64.93	\$0.37		2025-07-26	1.0.0
  GPT-5 mini (2025-08-07) (medium reasoning)	59.80	\$0.04		2025-08-07	1.7.0
  o3 (2025-04-16)	58.40	\$0.33		2025-07-26	1.0.0
  Qwen3-Coder 480B/A35B Instruct	55.40	\$		2025-08-02	1.0.0
  GLM-4.5 (2025-08-22)	54.20	\$0.30		2025-08-22	1.9.1

Figure source: <https://www.swebench.com/>



Thank You!

Yu Meng

University of Virginia

yumeng5@virginia.edu