LLM Agents

Yu Meng
University of Virginia
yumeng5@virginia.edu

Nov 13, 2024

UNIVERSITY of VIRGINIA

Join at
slido.com
#2959 913

Announcement

- Assignment 4 grades posted; reference answer released
- Contact Zhepei (tqf5qb@virginia.edu) if you have questions about your Assignment 4 grade
- Second guest lecture this Friday (11/15) – same policy as the first guest lecture
- We'll meet on Zoom (https://virginia.zoom.us/j/8397490876); no need to come to the classroom!
- We'll take attendance on Zoom
  - You'll get 1% participation credit for attending the guest lecture
  - Make sure your full name on Zoom matches your name on Canvas!
- You are encouraged to ask questions related to the talk!
  - You'll get another 1% participation credit if you ask a question (even if it does not get answered due to time constraints)
  - You can either ask directly during the talk or type your question in the Zoom chat (we count both), but we won't be using Slido for guest lectures

UNIVERSITY of VIRGINIA

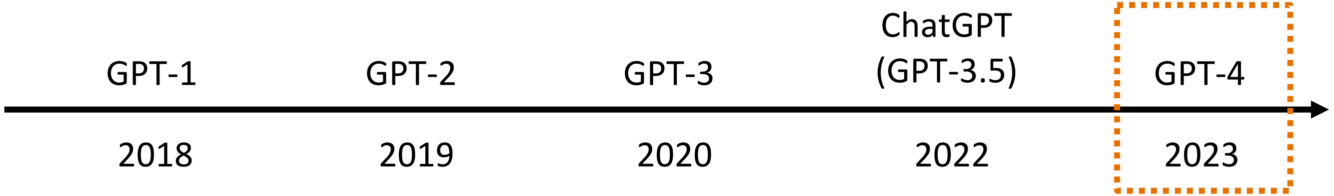
Join at
slido.com
#2959 913

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 and Neural Language Models
- Week 6-7: Language Modeling with Transformers (Pretraining + Fine-tuning)
- Week 8: Large Language Models (LLMs) & In-context Learning
- Week 9-10: Reasoning, Knowledge, and Retrieval-Augmented Generation (RAG)
- Week 11: LLM Alignment
- Week 12: Language Agents
- Week 13: Recap + Future of NLP
- Week 15 (after Thanksgiving): Project Presentations

(Recap) The Evolution of GPT Models: GPT-4

- 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
- **GPT-4: multimodality**



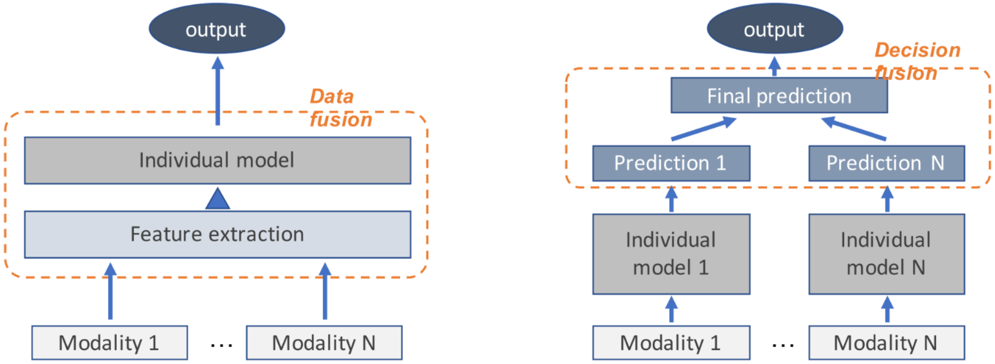
(Recap) Overview: Multimodal LLMs

- 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

(Recap) 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

Early fusion



Late fusion

Figure source: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9548039

(Recap) 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

(Recap) Learning Aligned Visual Representations

- Goal: learn a joint embedding space where images and their matching text descriptions are close together
- **CLIP** (Contrastive Language-Image Pretraining): predict the correct pairings of a batch of (image, text) training examples

Learning Transferable Visual Models From Natural Language Supervision

Alec Radford [*1] Jong Wook Kim [*1] Chris Hallacy [1] Aditya Ramesh [1] Gabriel Goh [1] Sandhini Agarwal [1]
Girish Sastry [1] Amanda Askell [1] Pamela Mishkin [1] Jack Clark [1] Gretchen Krueger [1] Ilya Sutskever [1]

Paper: https://arxiv.org/pdf/2103.00020

Join at
slido.com
#2959 913

(Recap) CLIP: Contrastive Pretraining

Maximize similarity between correct image-text pairs and minimize for incorrect ones

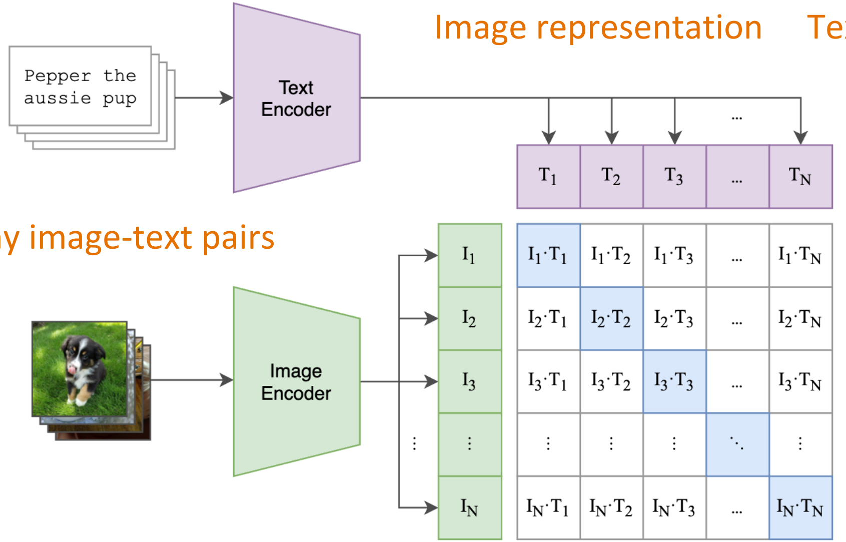
$$\mathcal{L}_{\text{CLIP}} = -\frac{1}{N}\sum_{i=1}^{N}\left(\log\frac{\exp(\cos(\boldsymbol{I}_i,\boldsymbol{T}_i)/\tau)}{\sum_{j=1}^{N}\exp(\cos(\boldsymbol{I}_i,\boldsymbol{T}_j)/\tau)} + \log\frac{\exp(\cos(\boldsymbol{T}_i,\boldsymbol{I}_i)/\tau)}{\sum_{j=1}^{N}\exp(\cos(\boldsymbol{T}_i,\boldsymbol{I}_j)/\tau)}\right)$$

Image representation

Text representation

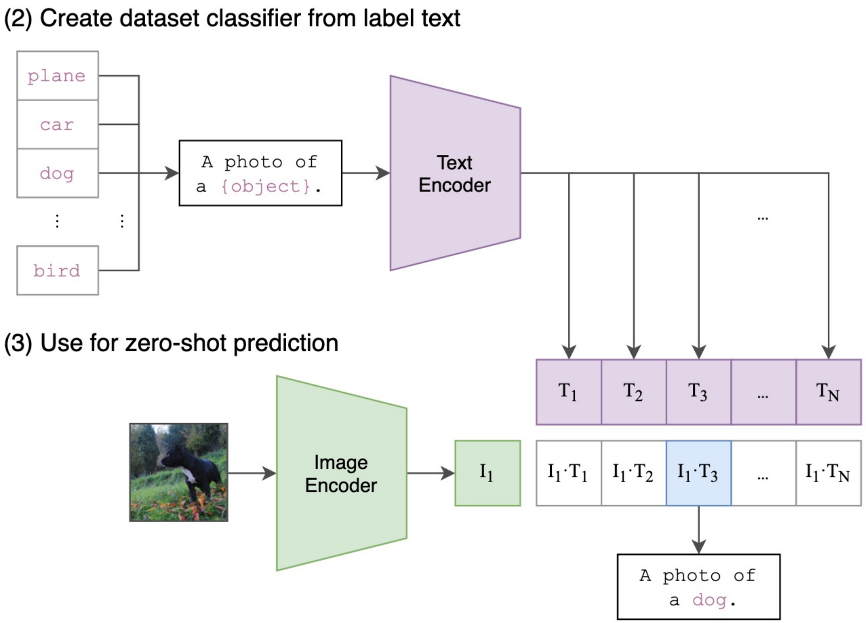
Learnable temperature

Many image-text pairs



CLIP: Zero-shot Generalization

After training, the text encoder/image encoder can embed the target class names/test images for zero-shot image classification



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

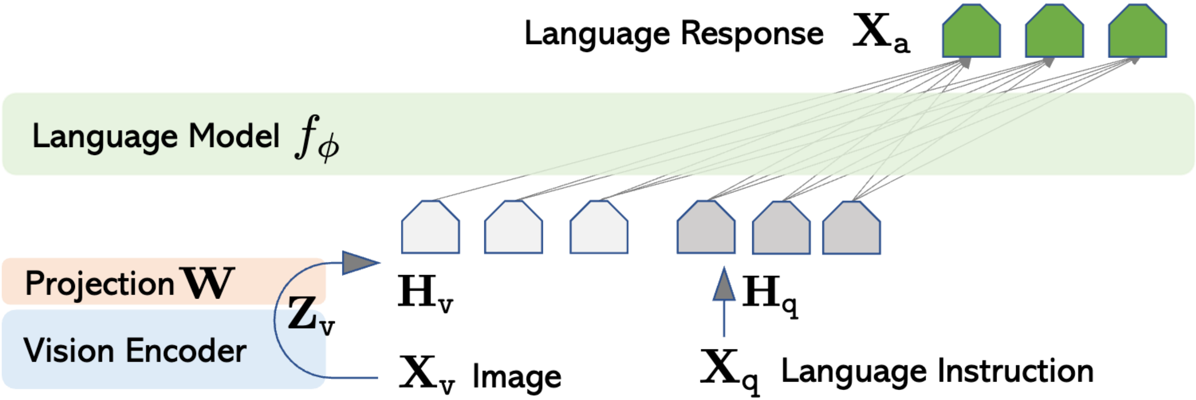
Haotian Liu[1*], Chunyuan Li[2*], Qingyang Wu[3], Yong Jae Lee[1]
[1]University of Wisconsin–Madison [2]Microsoft Research [3]Columbia University
https://llava-vl.github.io

Paper: https://arxiv.org/pdf/2304.08485

Join at
slido.com
#2959 913

LLaVA: Architecture

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

Join at
slido.com
#2959 913

User	Can you explain this meme in detail?
LLaVA	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

UNIVERSITY of VIRGINIA

Join at
slido.com
#2959 913

Further Reading on Multimodal LLMs

- Zero-Shot Text-to-Image Generation [Ramesh et al., 2021]
- Flamingo: a Visual Language Model for Few-Shot Learning [Alayrac et al., 2022]
- AudioLM: a Language Modeling Approach to Audio Generation [Borsos et al., 2022]
- Movie Gen: A Cast of Media Foundation Models [Polyak et al., 2024]

UNIVERSITY of VIRGINIA

Join at slido.com #2959 913

Agenda

- LLM Agent Overview
- Tool Usages
- Code Assistant

Join at
slido.com
#2959 913

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)

UNIVERSITY of VIRGINIA

Claude 3.5: Computer Use

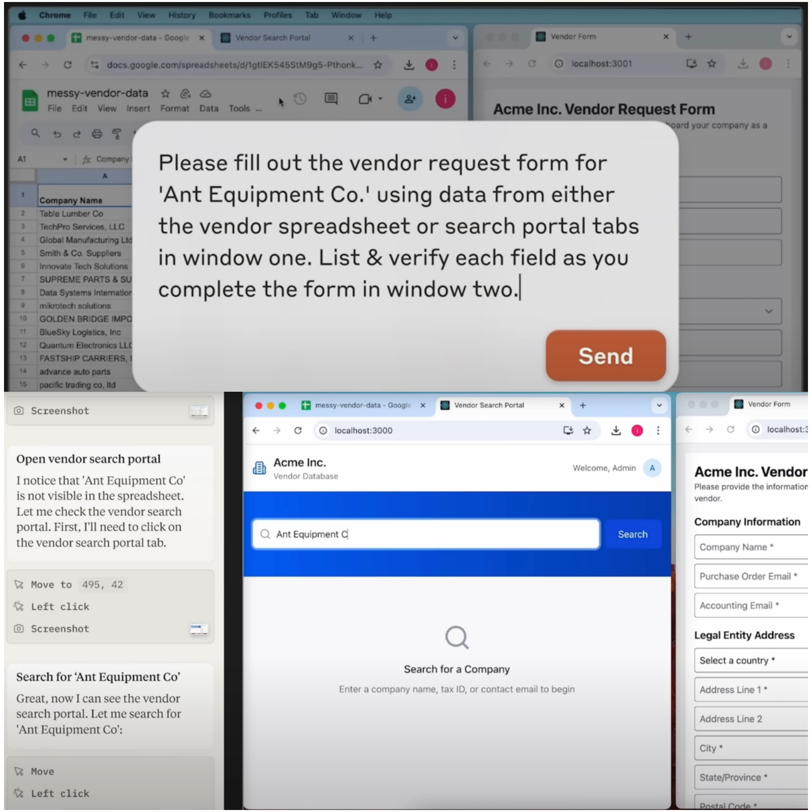


Figure source: https://www.anthropic.com/news/3-5-models-and-computer-use

UNIVERSITY of VIRGINIA

Join at
slido.com
#2959 913

WebShop: Language Agents for Online Shopping

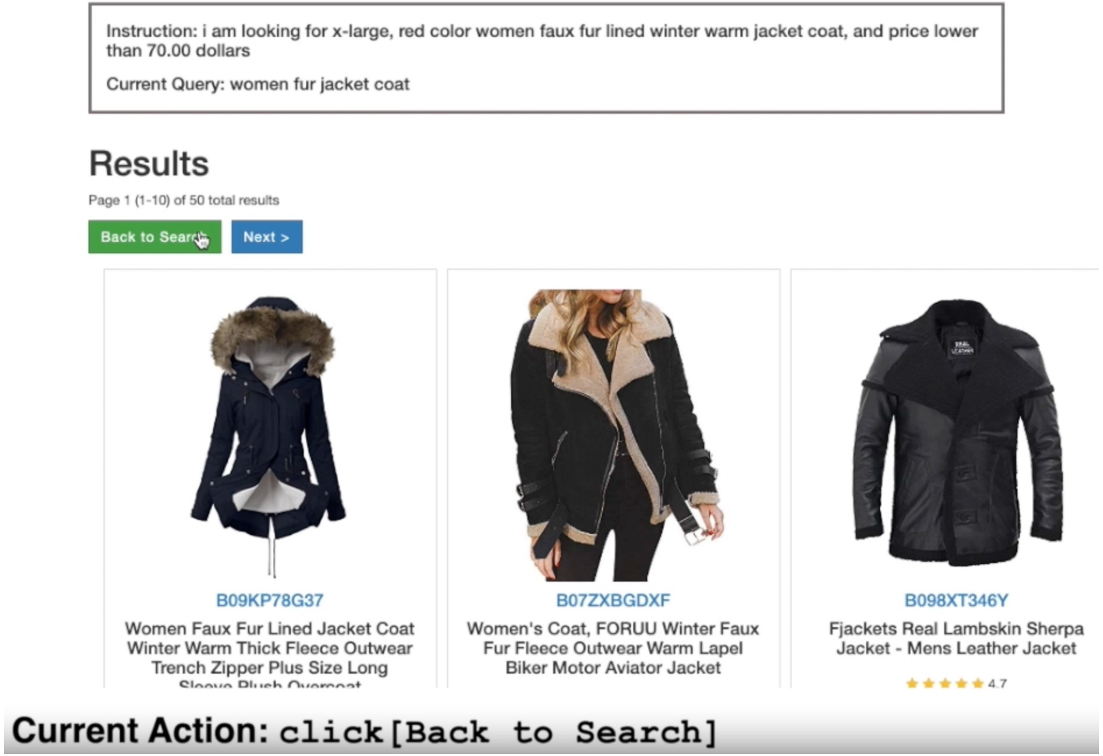


Figure source: https://webshop-pnlp.github.io/

Agenda

- LLM Agent Overview
- Tool Usages
- Code Assistant

UNIVERSITY of VIRGINIA

Join at
slido.com
#2959 913

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

Paper: https://arxiv.org/pdf/2302.04761

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

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:

In-context examples

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.

Generate API calls for new data

Input: $\mathbf{x}$

Output:

University of Virginia

Join at slido.com #2959 913

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

LM Dataset → **1** Sample API Calls → **2** Execute API Calls → **3** Filter API Calls → Final generation

$\mathbf{x}_{1:i-1}$ = Pittsburgh is also known as

c_i^1 = What other name is Pittsburgh known by?

r_i^1 = Steel City

$L_i(c_i^1 \rightarrow \text{Steel City})$
$< \boxed{\min(L_i(c_i^1 \rightarrow \varepsilon), L_i(\varepsilon))}$

Not performing the API call

- Filter out API calls which do not reduce the loss

c_i^2 = Which country is Pittsburgh in?

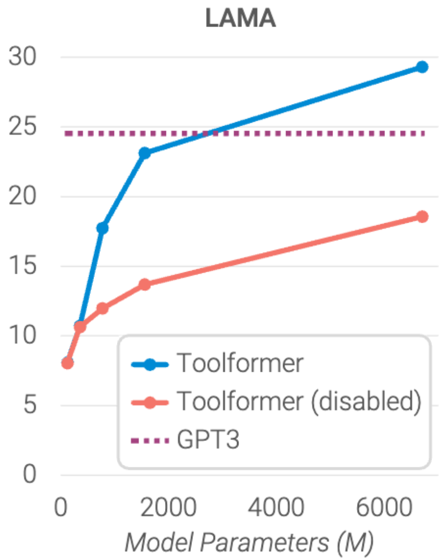
r_i^2 = United States

$$\boxed{\begin{aligned} &L_i(c_i^2 \rightarrow \text{United States}) \\ &> \min(L_i(c_i^2 \rightarrow \varepsilon), L_i(\varepsilon)) \end{aligned}}$$

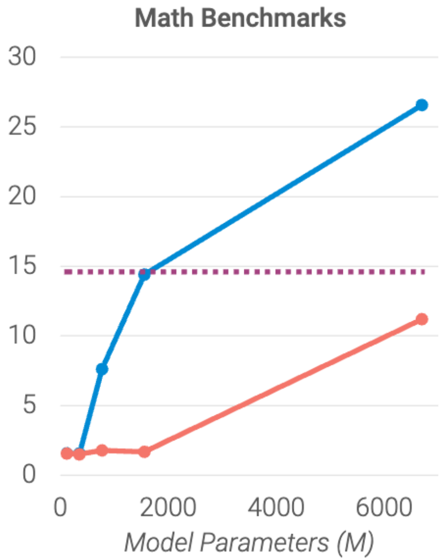
API call does not help reduce the loss

Tool Usage Ability vs. Model Scale

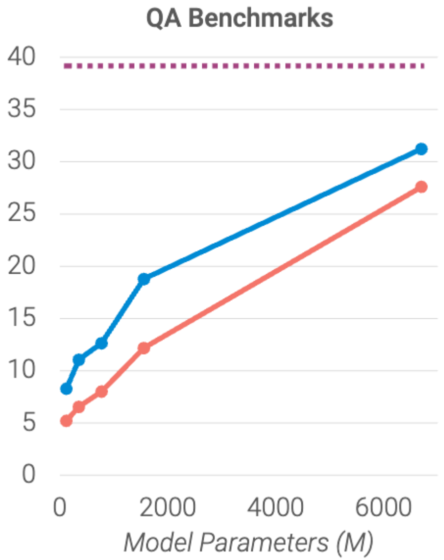
Larger models more effectively learn how to appropriately use tools



Wikipedia search



Calculator



Question answering

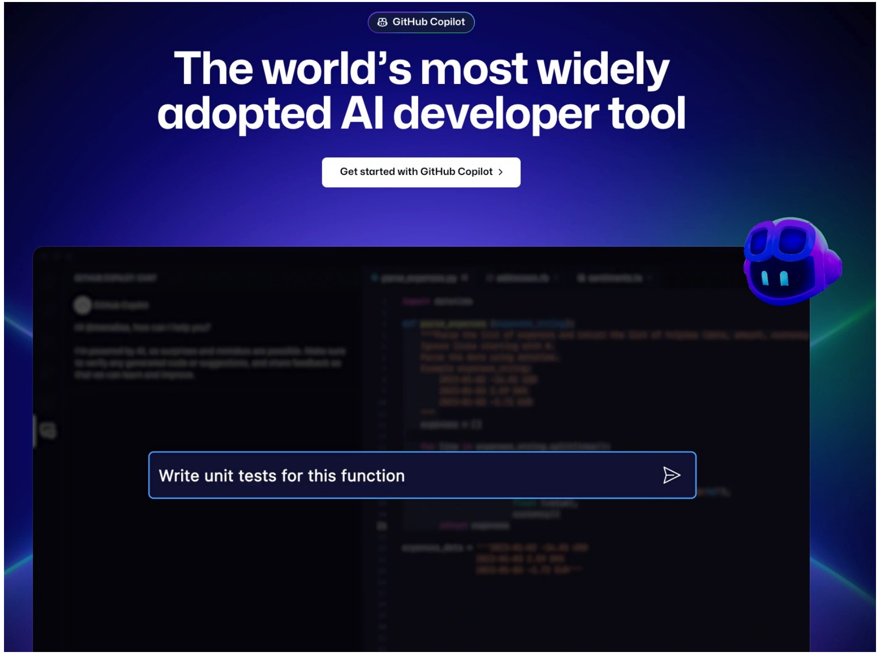
Agenda

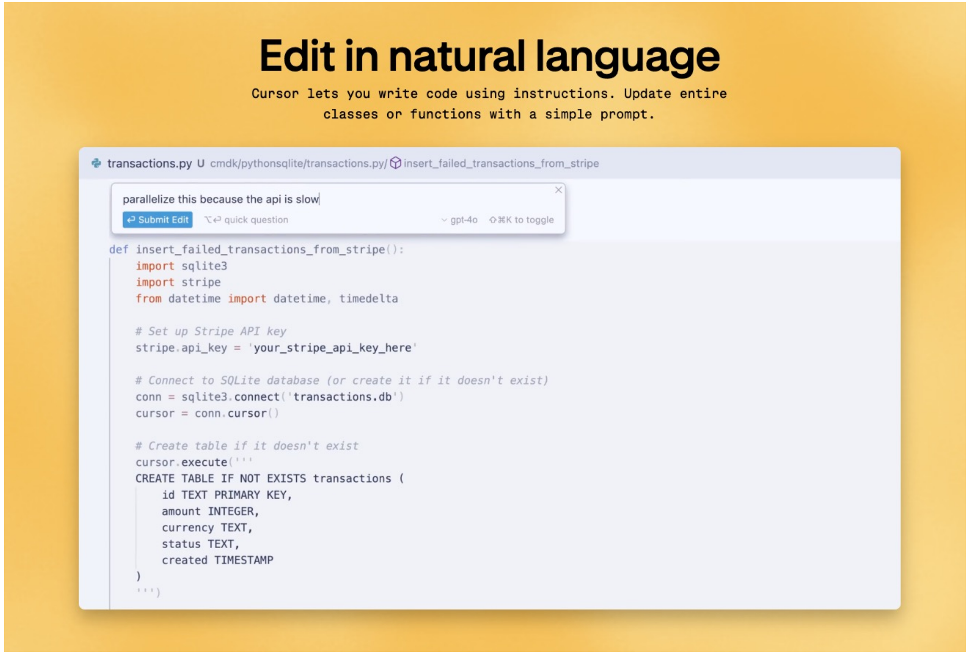
- LLM Agent Overview
- Tool Usages
- Code Assistant

UNIVERSITY of VIRGINIA

LLMs as Code Assistants

Join at
slido.com
#2959 913





https://github.com/features/copilot

https://www.cursor.com/

UNIVERSITY of VIRGINIA

Join at
slido.com
#2959 913

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

Paper: https://arxiv.org/pdf/2204.05999

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

```python
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

```python
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

```python
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

```python
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)
```

UNIVERSITY of VIRGINIA

Join at slido.com #2959 913

CodeLlama: LLMs for Code

- Can perform both code completion & infilling
- Support long input contexts (up to 16K tokens)
- Adopt instruction-tuning for improved safety and helpfulness

Code Llama: Open Foundation Models for Code

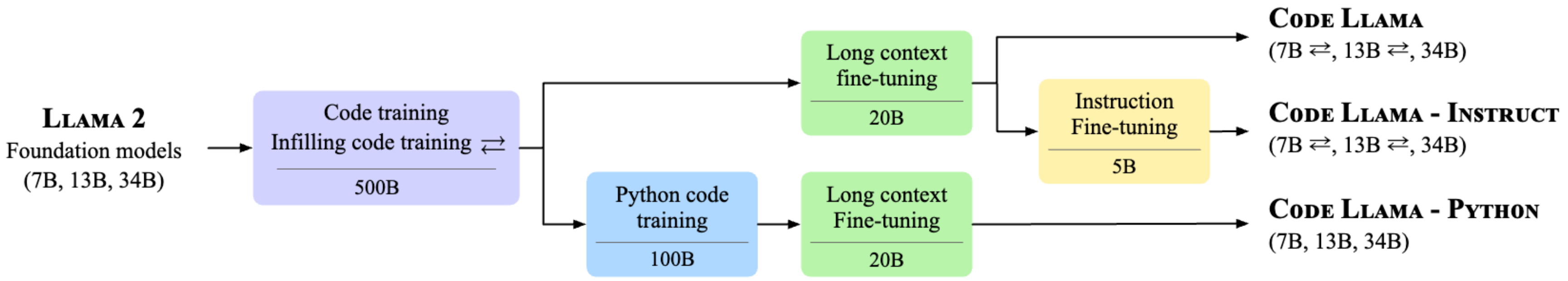
Baptiste Rozière†, Jonas Gehring†, Fabian Gloeckle†,*, Sten Sootla†, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi◇, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, Gabriel Synnaeve†

Meta AI

Paper: https://arxiv.org/pdf/2308.12950

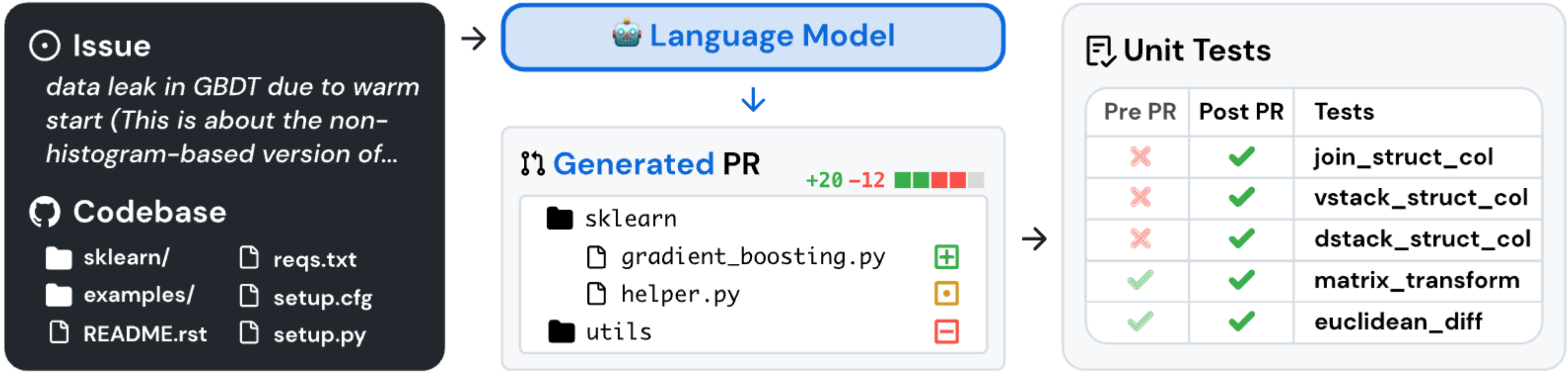
CodeLlama: Multi-stage Training

- Code Llama: a foundational model for code generation tasks
- Code Llama - Python: specialized for Python
- Code Llama - Instruct: fine-tuned with human instructions and synthetic data



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



Paper: https://arxiv.org/pdf/2310.06770

Thank You!

Yu Meng
University of Virginia
yumeng5@virginia.edu