Summary + Future of NLP

Yu Meng
University of Virginia
yumeng5@virginia.edu

Nov 18, 2024

Announcement: No Lectures Next Week

- No class next Monday (11/25)
- We'll hold an instructor office hour at the normal lecture time in this classroom
  - Assignment 5 (12/02 deadline)
  - Final project (presentation slides due 12/01; report due 12/13)

13	11/18	Summary & Future of NLP (I)			
	11/20	Summary & Future of NLP (II)			
	11/22	Guest Lecture: Yizhong Wang (University of Washington)			
14	11/25	Thanksgiving Recess (No Class)			
	11/27	Thanksgiving Recess (No Class)			
	11/29	Thanksgiving Recess (No Class)			**Project presentation slides due: 12/01 11:59pm (Guideline)**
15	12/02	Project Presentation			**Assignment 5 due: 12/02 11:59pm**
	12/04	Project Presentation			
	12/06	Project Presentation			**Project report due: 12/13 11:59pm (Guideline)**

Instructor office hour

Announcement: Final Project

- The week after Thanksgiving: three lectures (12/02, 12/04, 12/06) for project presentations
  - Held on Zoom: https://virginia.zoom.us/j/8397490876
  - We'll release a signup sheet after today's lecture; slots (8 slots per lecture) are first come, first served!
  - Regardless of your presentation date, you'll need to submit your slides by **12/01 11:59pm** (no late days allowed!)
  - Presentation guideline: https://docs.google.com/document/d/1JmTTx1ez9Zo6_zAA90TzVEq5ewGArR5UXRAm5sBcFTE/edit?usp=sharing
  - A good chance to receive feedback on your project (report due **12/13 11:59pm**)
  - Final report guideline: https://docs.google.com/document/d/1plsOtU6dvRF_EOVEvxrxfh0qXilHj0daWiTPWoTvFM4/edit?usp=sharing

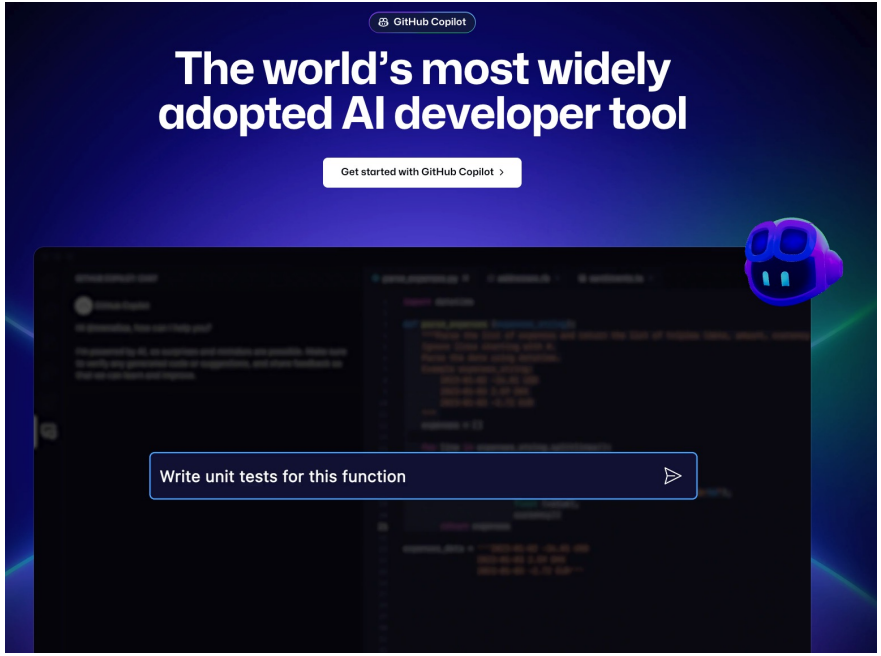
Overview of Course Contents

- 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: Summary + Future of NLP
- Week 15 (after Thanksgiving): Project Presentations

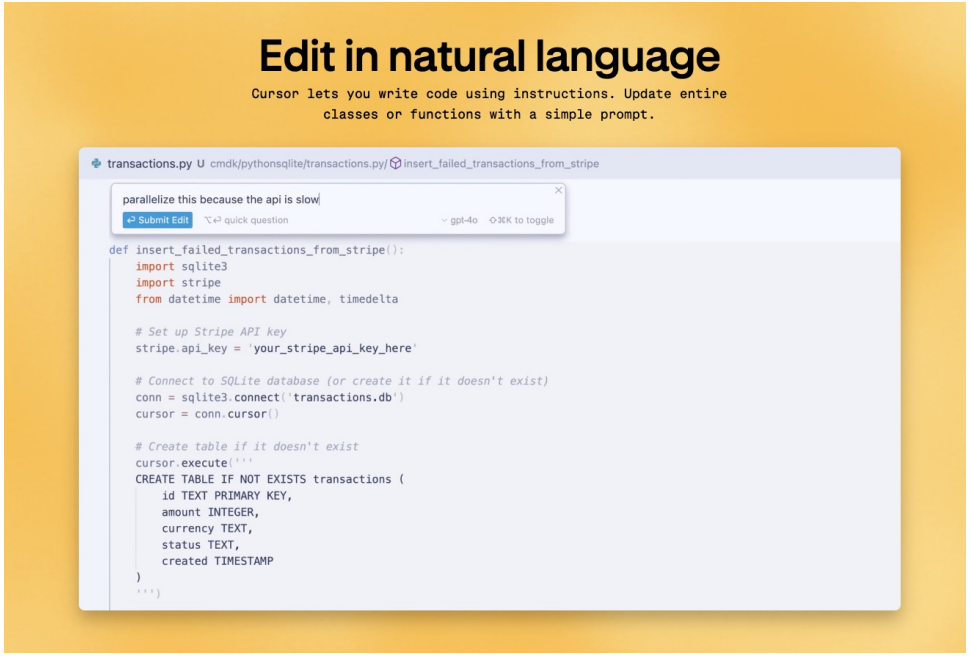
UNIVERSITY of VIRGINIA

Join at
slido.com
#4119 554

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 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

```
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)
```

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

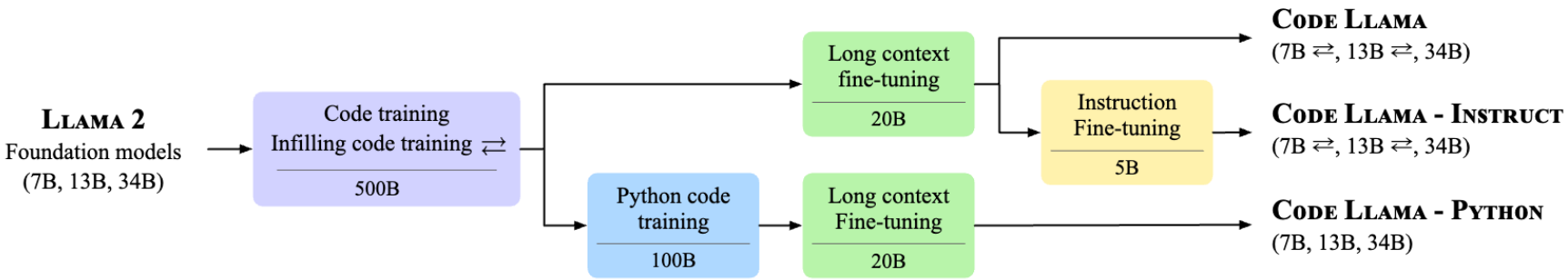
Baptiste Rozière[†], Jonas Gehring[†], Fabian Gloeckle[†,*], Sten Sootla[†], Itai Gat, Xiaoqing Ellen Tan, Yossi Adi[◇], Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre 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

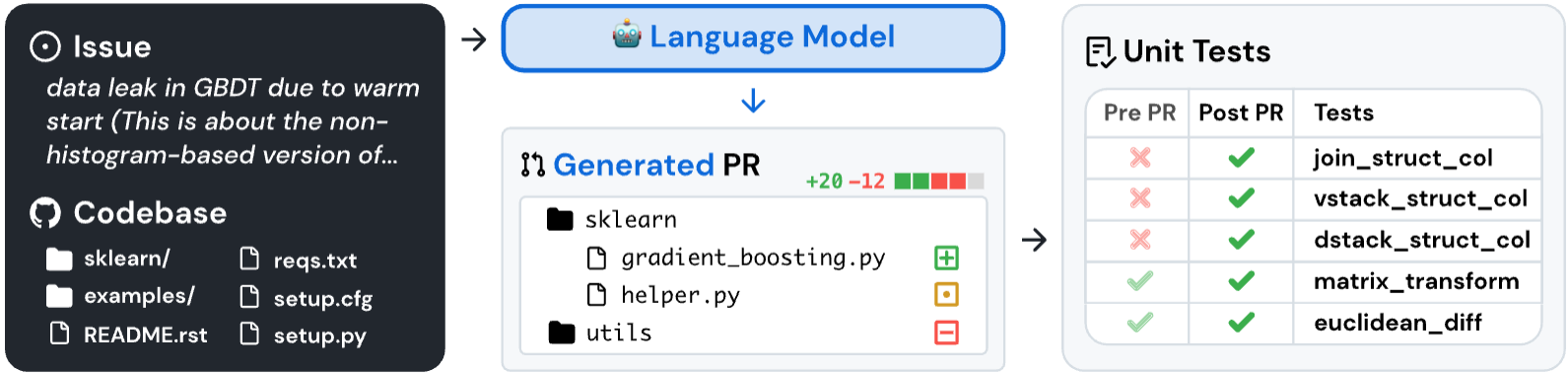
- Code Llama: a foundational model for code generation tasks
- Code Llama - Python: specialized for Python
- Code Llama - Instruct: fine-tuned with human instructions and synthetic data



Join at
slido.com
#4119 554

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 tests
- 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

Summary of Course Contents

- 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: Summary + Future of NLP

History of Language Models: N-gram LMs

- Language models started to be built with statistical methods
  - Sparsity
  - Poor generalization

N-gram Language Model: Simplified Assumption

- Challenge of language modeling: hard to keep track of all previous tokens!

$$p(\boldsymbol{x}) = \prod_{i=1}^{n} p(x_i | x_1, \ldots, x_{i-1})$$

Long context!
(Can we model long contexts at all?
Yes, but not for now!)

- Instead of keeping track of all previous tokens, assume the probability of a word is only dependent on the previous N−1 words

$$p(\boldsymbol{x}) = \prod_{i=1}^{n} p(x_i | x_1, \ldots, x_{i-1}) \approx \prod_{i=1}^{n} p(x_i | x_{i-N+1}, \ldots, x_{i-1})$$

N-gram assumption

Should N be larger or smaller?

Learning N-grams

- Probabilities can be estimated by frequencies (maximum likelihood estimation)!

$$p(x_i|x_{i-N+1},\ldots,x_{i-1}) = \frac{\#(x_{i-N+1},\ldots,x_{i-1},x_i)}{\#(x_{i-N+1},\ldots,x_{i-1})}$$

How many times (counts) the sequences occur in the corpus

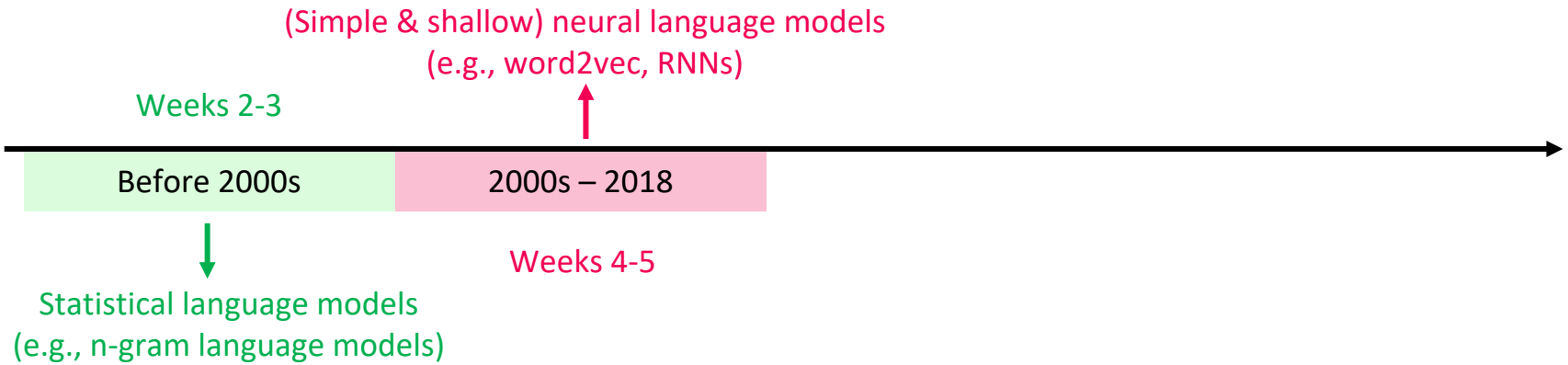
- Unigram: $p(x_i) = \frac{\#(x_i)}{\#(\text{all word counts in the corpus})}$
- Bigram: $p(x_i|x_{i-1}) = \frac{\#(x_{i-1},x_i)}{\#(x_{i-1})}$
- Trigram: $p(x_i|x_{i-2},x_{i-1}) = \frac{\#(x_{i-2},x_{i-1},x_i)}{\#(x_{i-2},x_{i-1})}$

N-gram Properties

- As N becomes larger
  - Better modeling of word correlations (incorporating more contexts)
  - Sparsity increases
- The number of possible N-grams (parameters) grows exponentially with N!
  - Suppose vocabulary size = 10K words
  - Possible unigrams = 10K
  - Possible bigrams = (10K)^2 = 100M
  - Possible trigrams = (10K)^3 = 1T
  - …

History of Language Models: Neural LMs

- The introduction of neural networks into language models mitigated sparsity and improved generalization
  - Neural networks for language models were small-scale and inefficient for a long time
  - Task-specific architecture designs required for different NLP tasks
  - These language models were trained on individual NLP tasks as task-specific solvers



Distributional Hypothesis

- Words that occur in similar contexts tend to have similar meanings
- A word's meaning is largely defined by the company it keeps (its context)
- Example: suppose we don't know the meaning of "Ong choy" but see the following:
  - Ong choy is delicious **sautéed with garlic**
  - Ong choy is superb **over rice**
  - … ong choy **leaves** with **salty** sauces
- And we've seen the following contexts:
  - … spinach **sautéed with garlic over rice**
  - … chard stems and **leaves** are **delicious**
  - … collard greens and other **salty** leafy greens
- Ong choy = water spinach!

Example source: https://web.stanford.edu/~jurafsky/slp3/slides/vectorsemantics2024.pdf

Word Embeddings: General Idea

- Learn dense vector representations of words based on distributional hypothesis
- Semantically similar words (based on context similarity) will have similar vector representations
- **Embedding**: a mapping that takes elements from one space and represents them in a different space

$$\boldsymbol{v}_{\text{to}} = [1, 0, 0, 0, 0, 0, \ldots]$$
$$\boldsymbol{v}_{\text{by}} = [0, 1, 0, 0, 0, 0, \ldots]$$
$$\boldsymbol{v}_{\text{that}} = [0, 0, 1, 0, 0, 0, \ldots]$$
$$\boldsymbol{v}_{\text{good}} = [0, 0, 0, 1, 0, 0, \ldots]$$
$$\boldsymbol{v}_{\text{nice}} = [0, 0, 0, 0, 1, 0, \ldots]$$
$$\boldsymbol{v}_{\text{bad}} = [0, 0, 0, 0, 0, 1, \ldots]$$

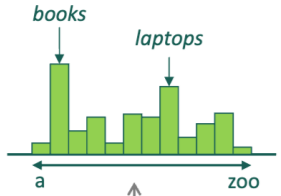
2D visualization of a word embedding space

Figure source: https://web.stanford.edu/~jurafsky/slp3/6.pdf

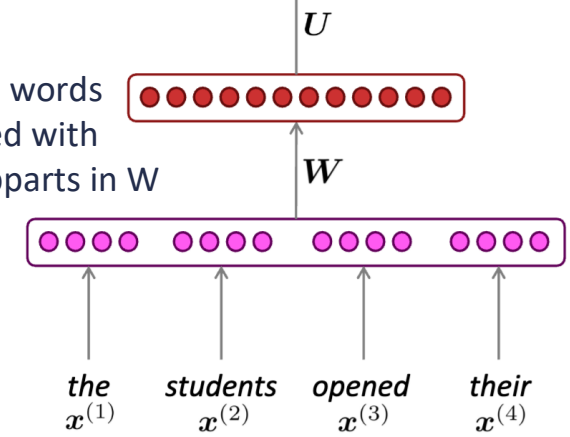
Recurrent Neural Network (RNN) Overview

A neural language model that can process inputs of arbitrary lengths

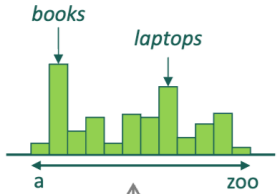
Simple neural language model



Different words multiplied with different subparts in W



Recurrent neural language model



Reuse the same weights for all words

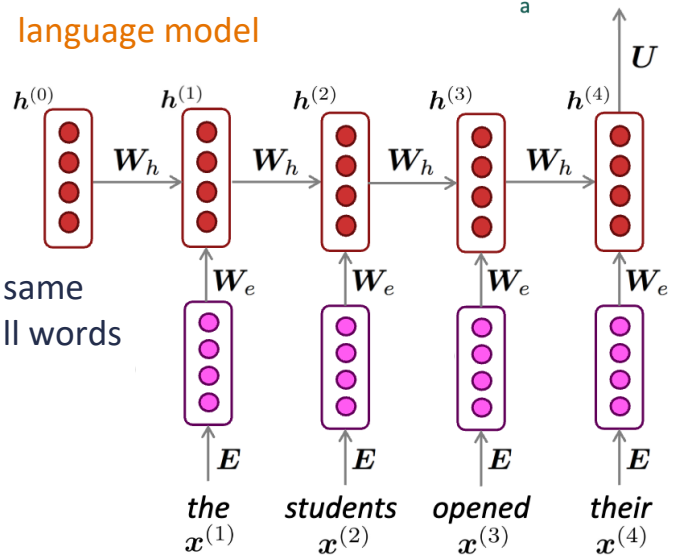


Figure source: https://web.stanford.edu/class/cs224n/slides/cs224n-spr2024-lecture05-rnnlm.pdf

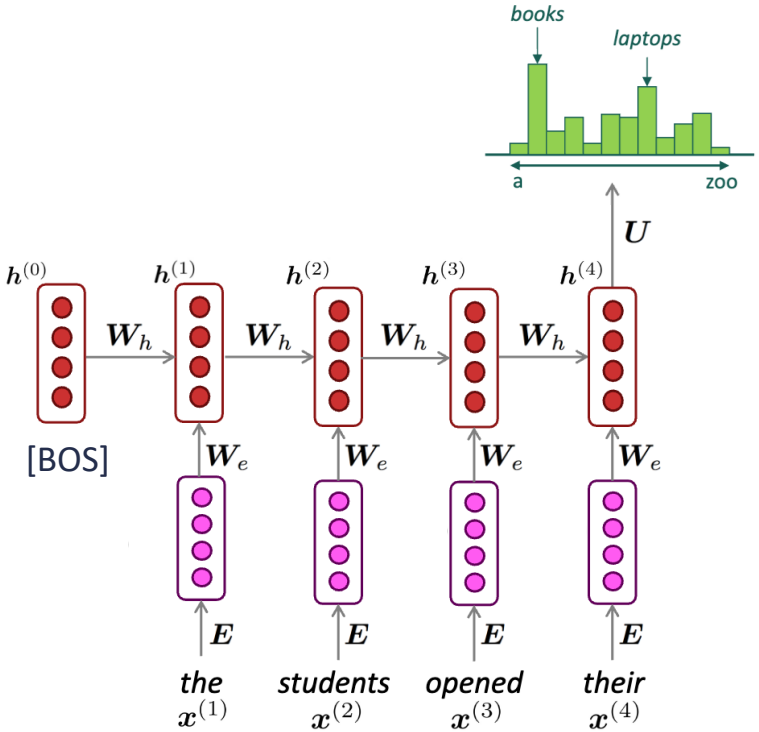
RNN Computation

- Hidden states in RNNs are computed based on
  - The hidden state at the previous step (memory)
  - The word embedding at the current step
- Parameters:
  - $\boldsymbol{W}_h$: weight matrix for the recurrent connection
  - $\boldsymbol{W}_e$: weight matrix for the input connection

$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e \boldsymbol{x}^{(t)}\right)$$

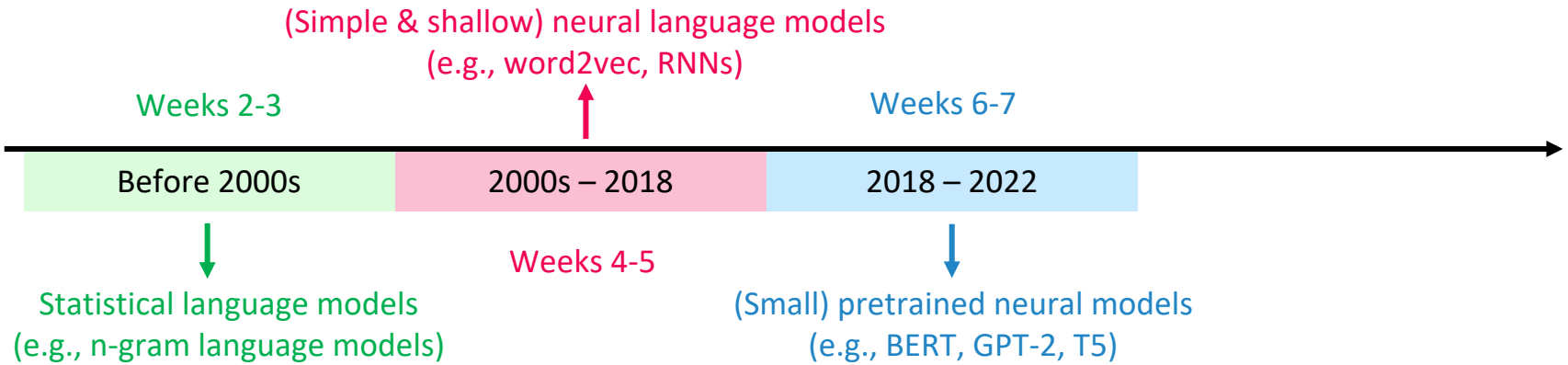
Hidden states at the previous word (time step)

Word embedding of the current word (time step)



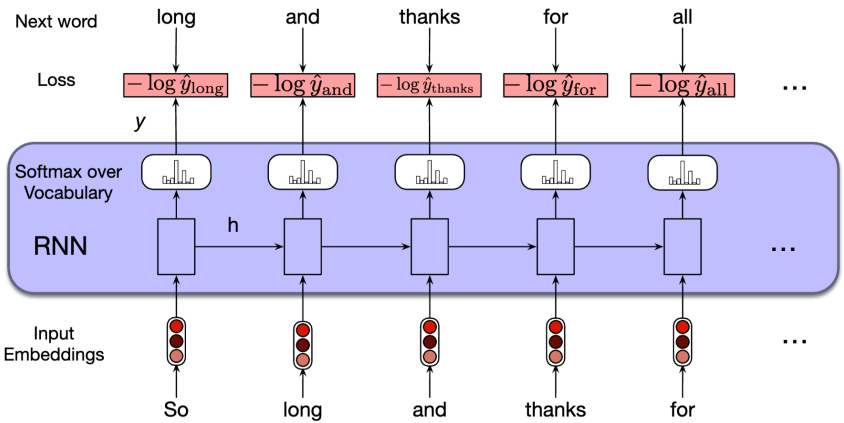
History of Language Models: Transformer LMs

- Transformer became the dominant architecture for language modeling; scaling up model sizes and (pretraining) data enabled significant generalization ability
  - Transformer demonstrated striking scalability and efficiency in sequence modeling
  - One pretrained model checkpoint fine-tuned to become strong task-specific models
  - Task-specific fine-tuning was still necessary



Transformer vs. RNN

RNN
(recurrent computations)



Transformer
(self-attention computations)

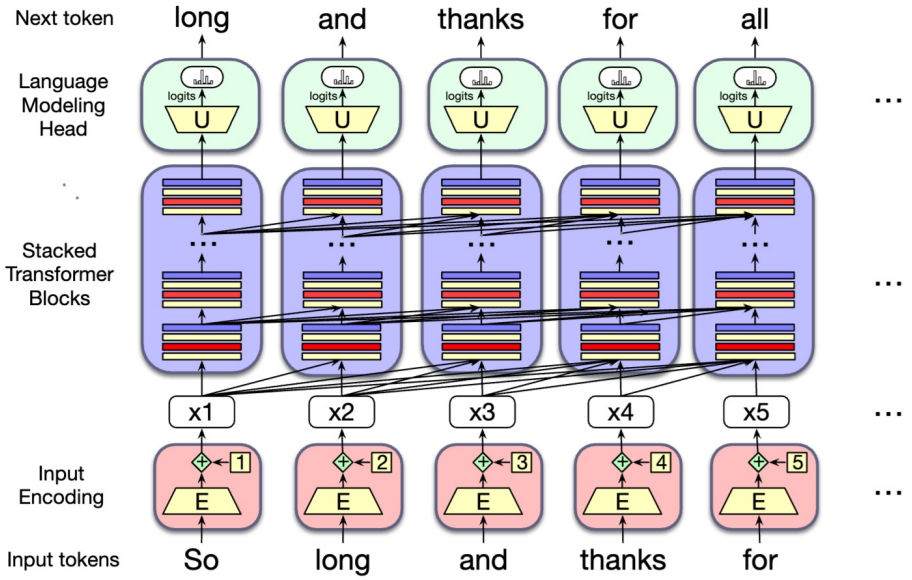


Figure source: https://web.stanford.edu/~jurafsky/slp3/9.pdf

Transformer Block

- Modules in Transformer layers:
  - Multi-head self-attention
  - Layer normalization (LayerNorm)
  - Feedforward network (FFN)
  - Residual connection

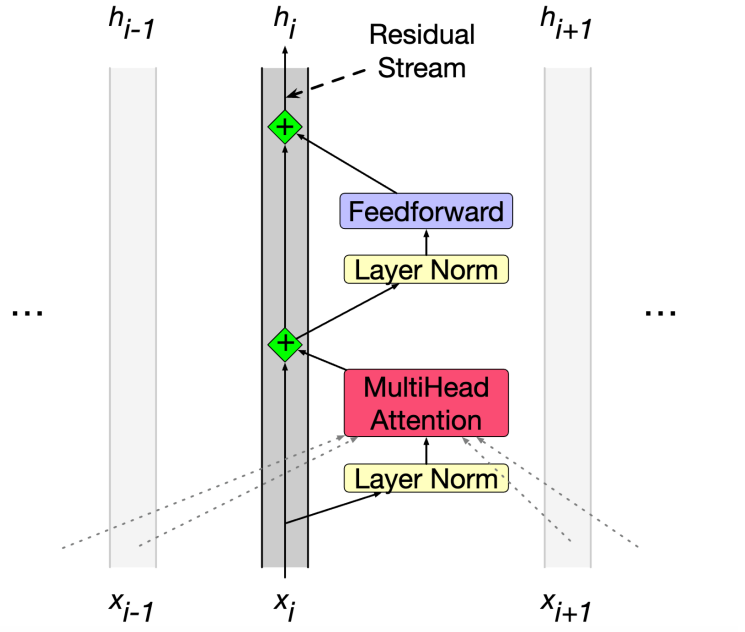


Figure source: https://web.stanford.edu/~jurafsky/slp3/9.pdf

Self-Attention

Derive the center word representation as a weighted sum of context representations!

Center word representation — Context word representation

$$\boldsymbol{a}_i = \sum_{x_j \in \boldsymbol{x}} \alpha_{ij} \boldsymbol{x}_j, \quad \sum_{x_j \in \boldsymbol{x}} \alpha_{ij} = 1$$

Attention score $i \rightarrow j$, summed to 1

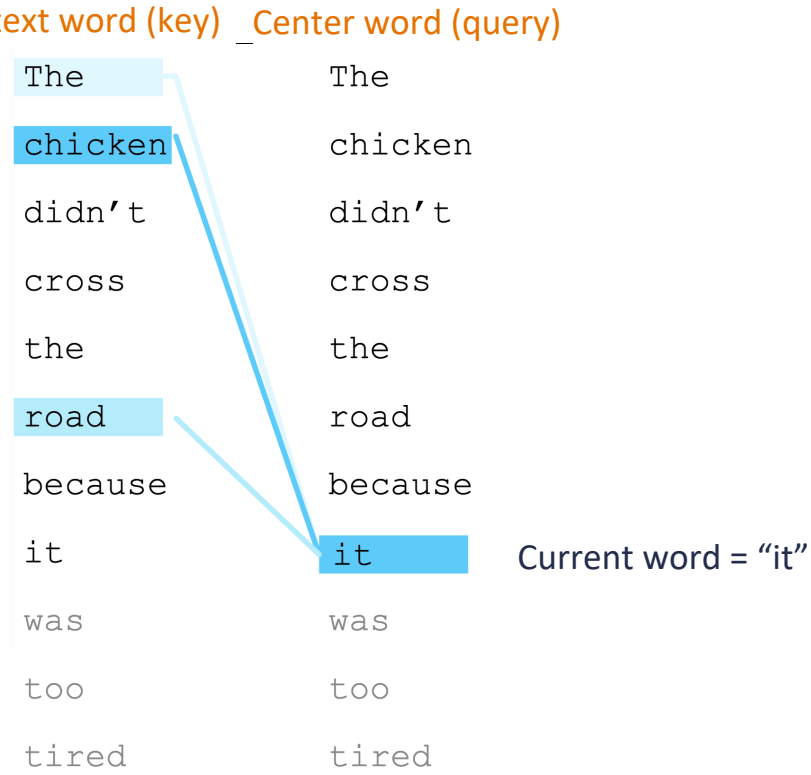


Figure source: https://web.stanford.edu/~jurafsky/slp3/9.pdf

Pretraining + Fine-Tuning

- Pretraining: trained with pretext tasks on large-scale text corpora
- Fine-tuning (continue training): adjust the pretrained model's parameters with fine-tuning data
- Fine-tuning data can have different forms:
  - Task-specific labeled data (e.g., sentiment classification, named entity recognition)
  - (Multi-turn) dialogue data (i.e., instruction tuning)

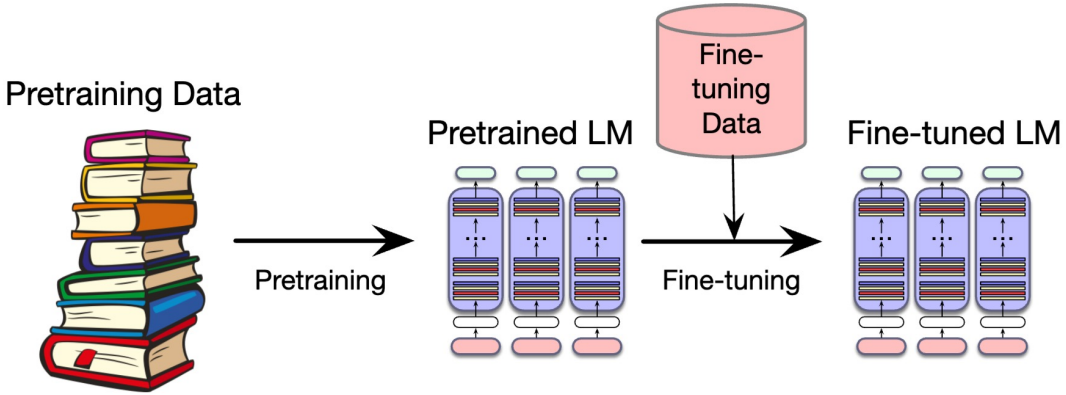


Figure source: https://web.stanford.edu/~jurafsky/slp3/10.pdf

Pretraining: Multi-Task Learning

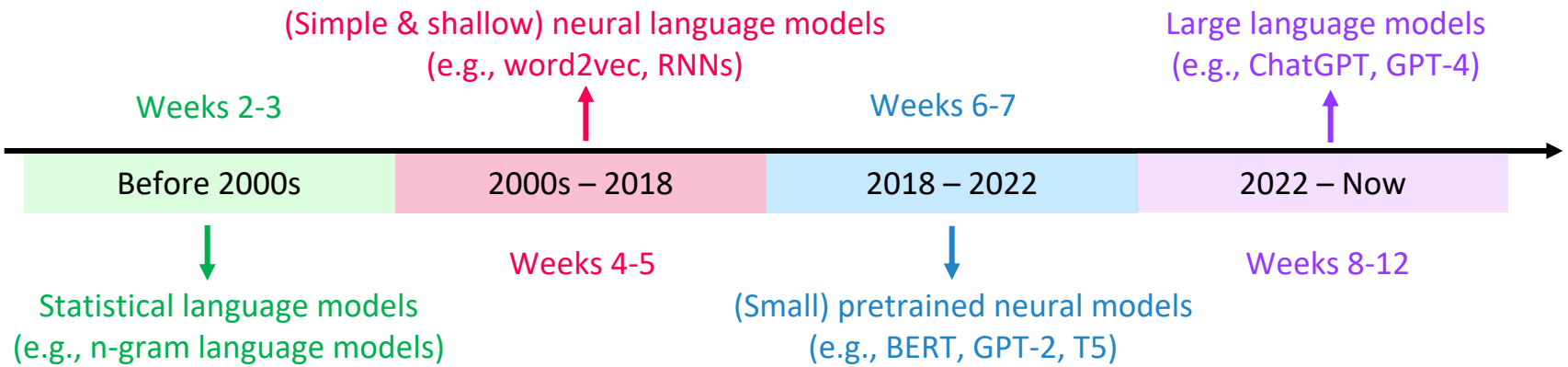
- In my free time, I like to **{run, banana}** (*Grammar*)
- I went to the zoo to see giraffes, lions, and **{zebras, spoon}** (*Lexical semantics*)
- The capital of Denmark is **{Copenhagen, London}** (*World knowledge*)
- I was engaged and on the edge of my seat the whole time. The movie was **{good, bad}** (*Sentiment analysis*)
- The word for "pretty" in Spanish is **{bonita, hola}** (*Translation*)
- 3 + 8 + 4 = **{15, 11}** (*Math*)
- …

Examples from: https://docs.google.com/presentation/d/1hQUd3pF8_2Gr2Obc89LKjmHL0DlH-uof9M0yFVd3FA4/edit#slide=id.g28e2e9aa709_0_1

Join at
slido.com
#4119 554

History of Language Models

- Generalist large language models (LLMs) became the universal task solvers and replaced task-specific language models
  - Real-world NLP applications are usually multifaceted (require composite task abilities)
  - Tasks are not clearly defined and may overlap
  - Single-task models struggle to handle complex tasks



In-context Learning

- In-context learning is a type of few-shot learning
  - User provides a few examples of input-output pairs in the prompt
  - The model uses given examples to predict the output for new, similar inputs
- First studied in the GPT-3 paper
- No model parameter updates

Few-shot

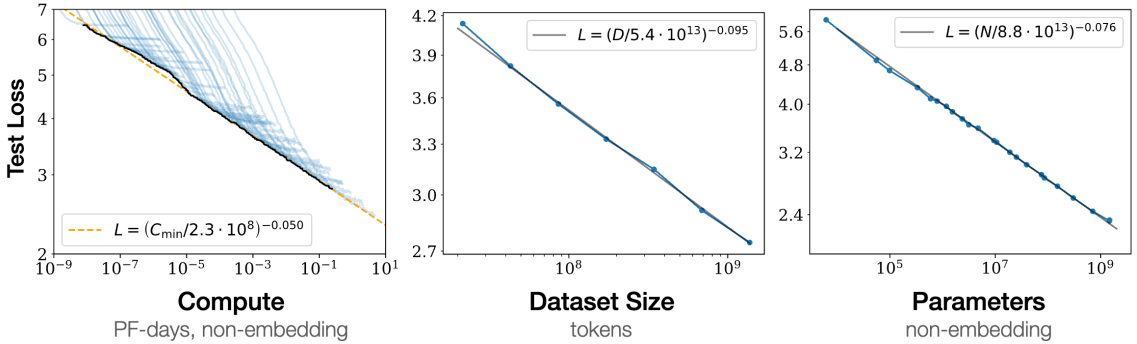
In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
1  Translate English to French:        ← task description
2  sea otter => loutre de mer          ← examples
3  peppermint => menthe poivrée        ←
4  plush girafe => girafe peluche      ←
5  cheese =>                           ← prompt
```

Figure source: https://arxiv.org/pdf/2005.14165

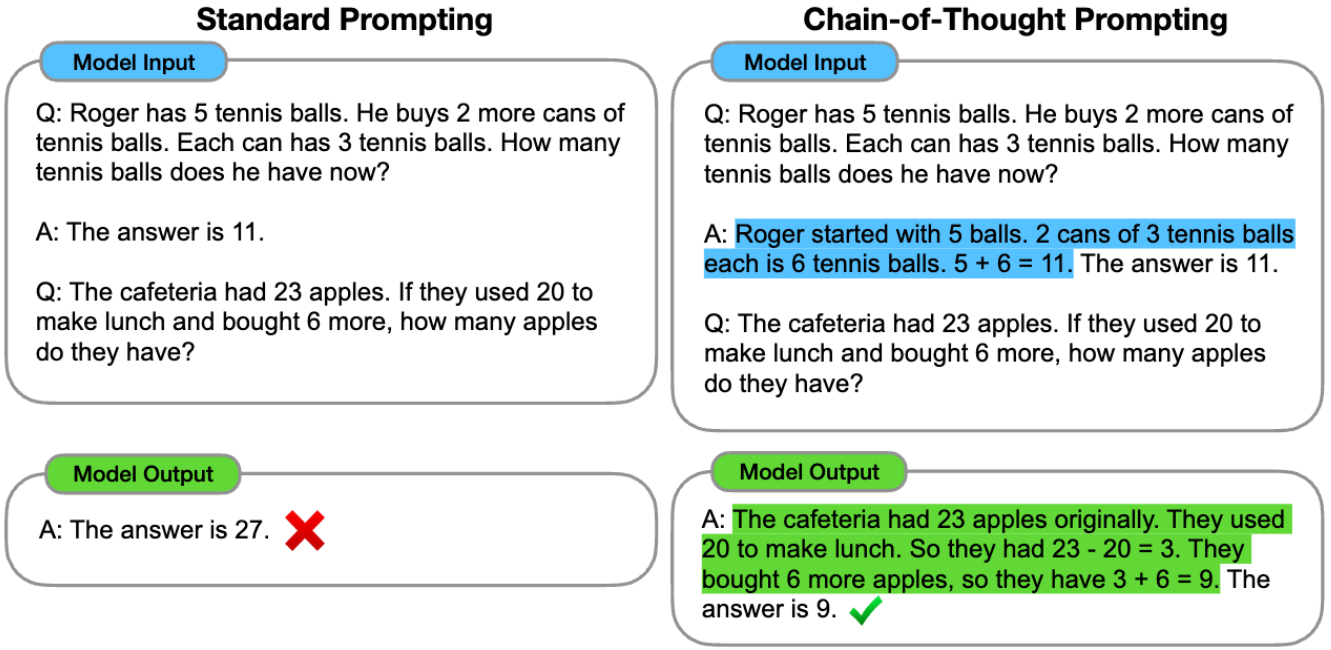
Scaling Laws of LLMs

- (Pretrained) LLM performance is mainly determined by 3 factors
  - Model size: the number of parameters
  - Dataset size: the amount of training data
  - Compute: the amount of floating point operations (FLOPs) used for training
- Scaling up LLMs involves scaling up the 3 factors
  - Add more parameters (adding more layers or having more model dimensions or both)
  - Add more data
  - Train for more iterations



Paper: https://arxiv.org/pdf/2001.08361

Standard Prompting vs. CoT Prompting

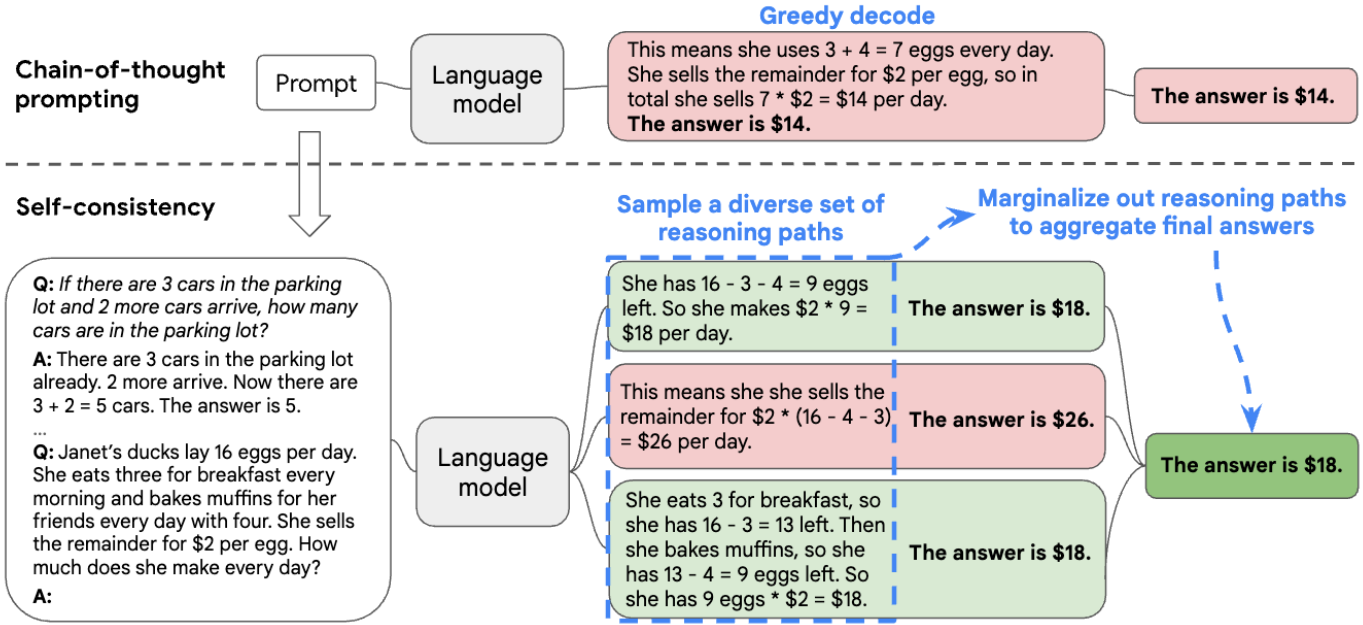


Paper: https://arxiv.org/pdf/2201.11903

Join at
slido.com
#4119 554

Self-consistency CoT

Intuition: if multiple different ways of thinking lead to the same answer, one has greater confidence that the final answer is correct



Paper: https://arxiv.org/pdf/2203.11171

Prompting LMs: Parametric Knowledge

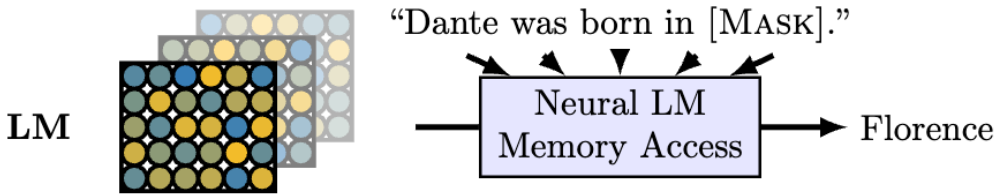
- LMs have learned from a lot of facts in their pretraining data
- LMs can be directly prompted to generate answers to factoid questions (Closed-book QA setting)
- Example:

$P(w|$`Q: Who wrote the book ‘‘The Origin of Species"? A:`$)$ prompt

- Since prompting LLMs only relies on the information stored within the parameters of the model itself, this kind of knowledge is called **parametric knowledge**

Language Model as Knowledge Bases

- **Acquisition**: LM's knowledge is derived from the vast amount of pretraining data
- **Access**: information is accessed through natural language prompts
- **Update/maintenance**: re-training/fine-tuning the model with new data
- **Pros**:
  - Handle a wide range of natural language queries with contextual understanding
  - Generalize to unseen queries not seen during training
- **Cons**:
  - May produce incorrect/outdated information
  - Lack interpretability/transparency



Paper: https://arxiv.org/pdf/1909.01066

Hallucination

- **Hallucination**: LM generates information that is factually incorrect, misleading, or fabricated, even though it may sound plausible or convincing
- Why does hallucination happen?
  - Limited knowledge: LLMs are trained on finite datasets, which don't have access to all possible information; when asked about topics outside their training data, they may generate plausible-sounding but incorrect responses
  - Overgeneralization: LLMs may apply patterns they've learned from one context to another where they don't apply, leading to incorrect conclusions
  - Lack of common sense: While LLMs can process and generate human-like text, they often lack the ability to apply commonsense reasoning to their outputs
  - …

Join at
slido.com
#4119 554

Retrieval-Augmented Generation

- Use a retriever to obtain relevant documents to the query from an external text collection
- Use LLMs to generate answers given the documents and a prompt

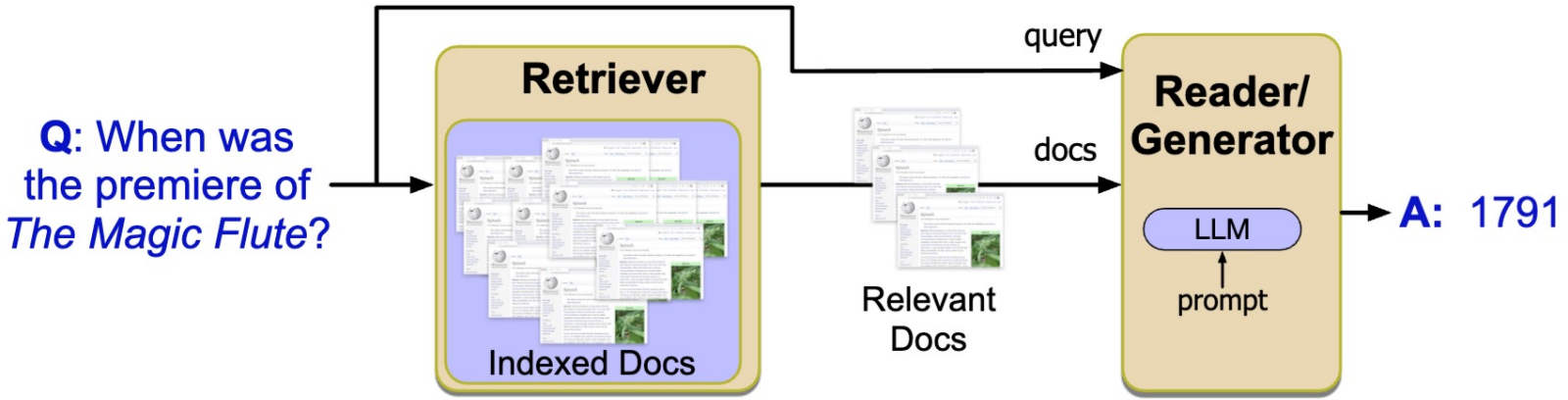


Figure source: https://web.stanford.edu/~jurafsky/slp3/14.pdf

Dense Retrieval

- Motivation: sparse retrieval (e.g., TF-IDF) relies on the exact overlap of words between the query and document without considering semantic similarity
- Solution: use a language model to obtain (dense) distributed representations of query and document
- The retriever language model is typically a small text encoder model (e.g., BERT)
  - Retrieval is a natural language understanding task
  - Encoder-only models are more efficient than LLMs for this purpose
- Both query and document representations are computed by text encoders

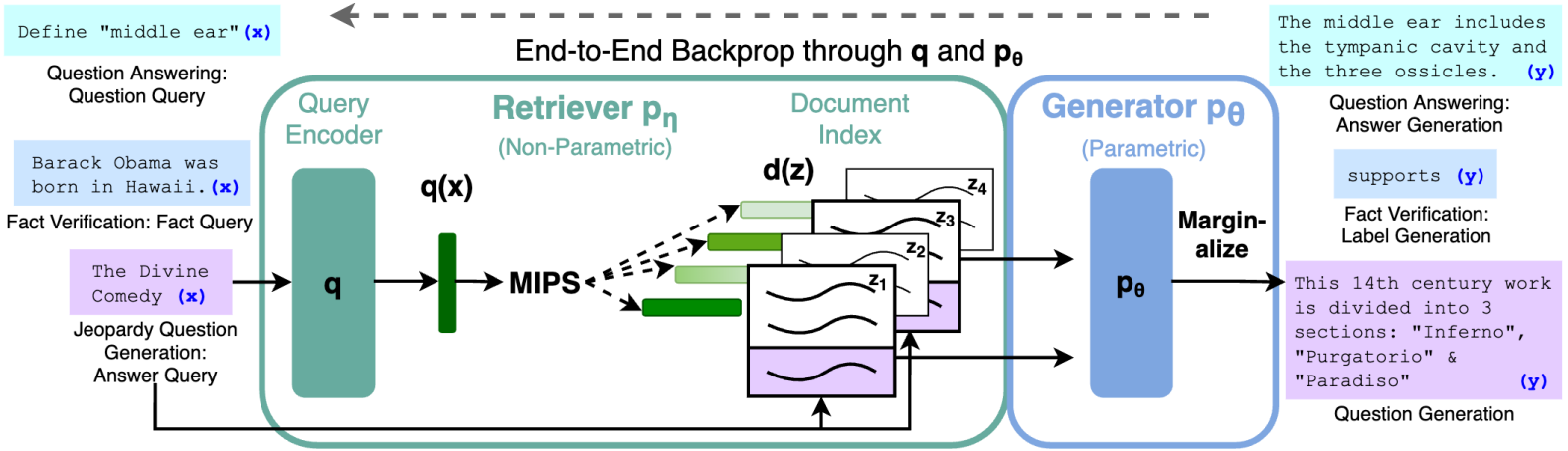
RAG: A Latent Variable Model

The retrieved documents are treated as latent variables (z) for generation

$$p(\boldsymbol{y}|\boldsymbol{x}) = \sum_{\boldsymbol{z}\in\mathcal{D}} p(\boldsymbol{z}|\boldsymbol{x})p(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{z})$$

Retrieve document (z) based on query (x)

Generate answer (y) based on retrieved docs (z) and query (x)



Paper: https://arxiv.org/pdf/2005.11401

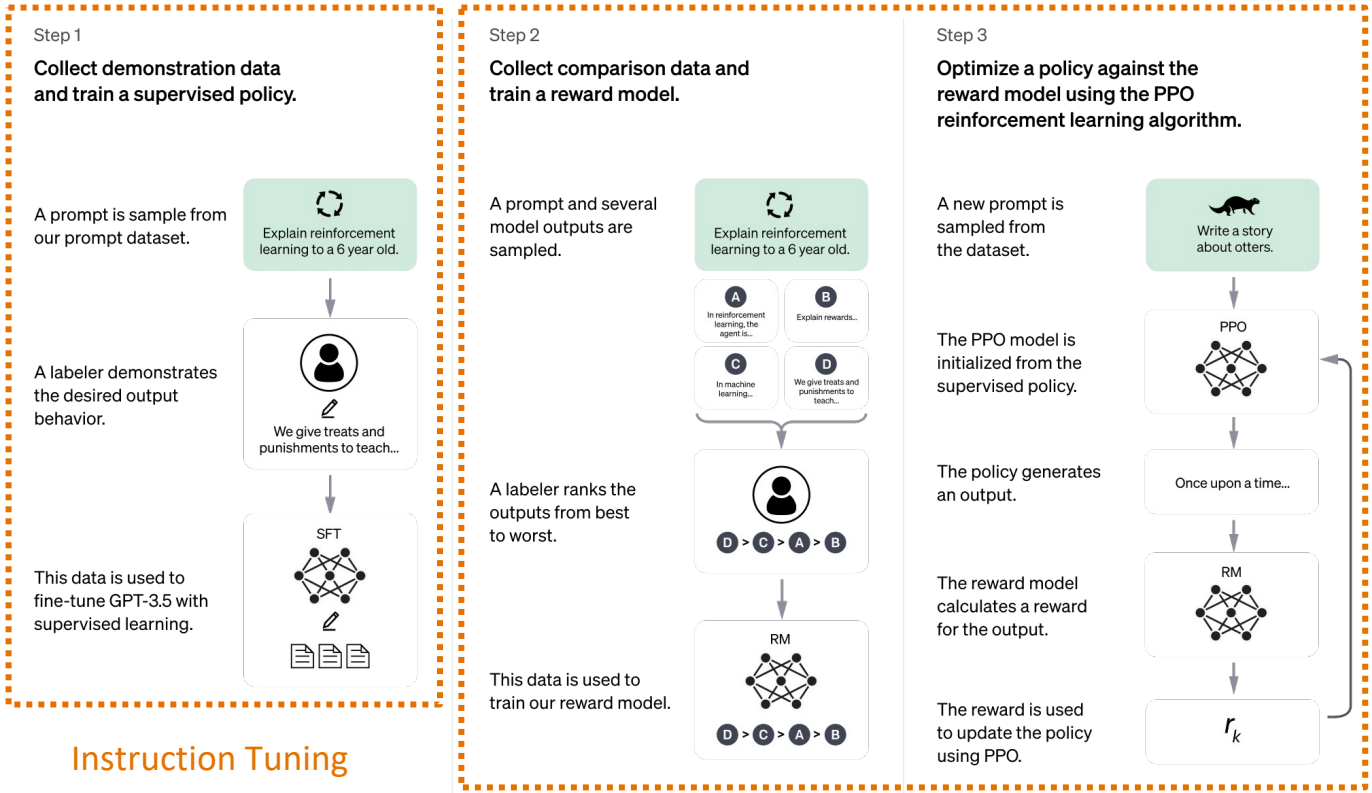
Language Model Alignment

- Ensure language models behaviors are aligned with human values and intent
- "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

Paper: https://arxiv.org/pdf/2112.00861

Join at
slido.com
#4119 554

Language Model Alignment Techniques



Instruction Tuning

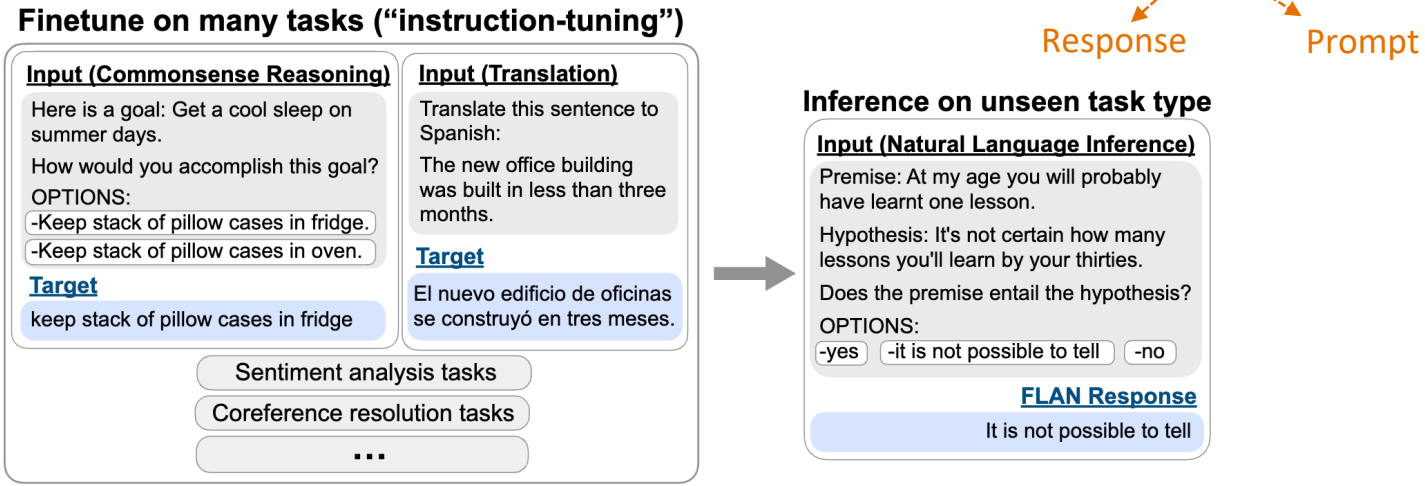
Reinforcement Learning from Human Feedback (RLHF)

Figure source: https://openai.com/index/chatgpt/

Instruction Tuning

- **Input**: task description
- **Output**: expected response or solution to the task
- Train LLMs to generate response tokens given prompts $\min_{\theta} -\log p_{\theta}(\boldsymbol{y}|\boldsymbol{x})$

Response Prompt



Paper: https://arxiv.org/pdf/2109.01652

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

RLHF Workflow

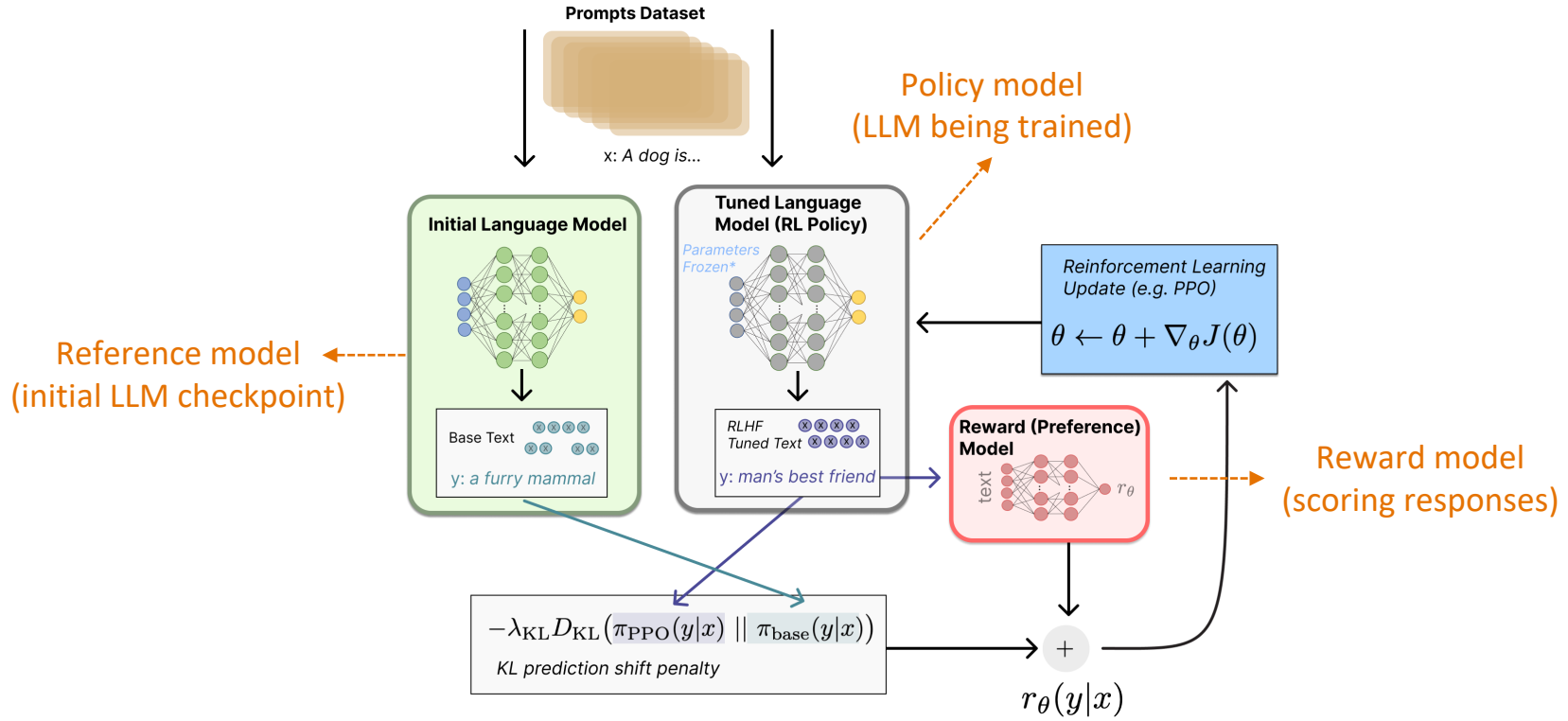


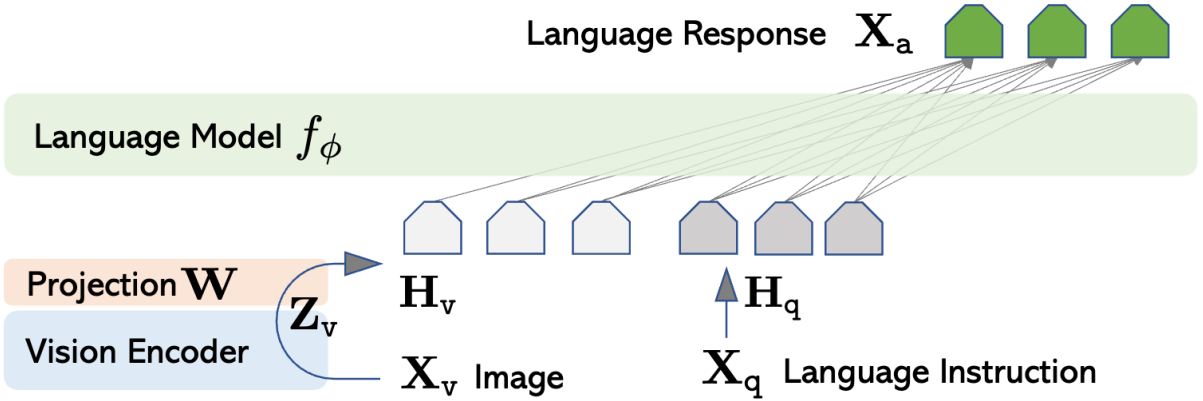
Figure source: https://huggingface.co/blog/rlhf

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

Visual Instruction Tuning: LLaVA

- 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**

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)

Tool Usages with LLMs: Toolformer

- 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

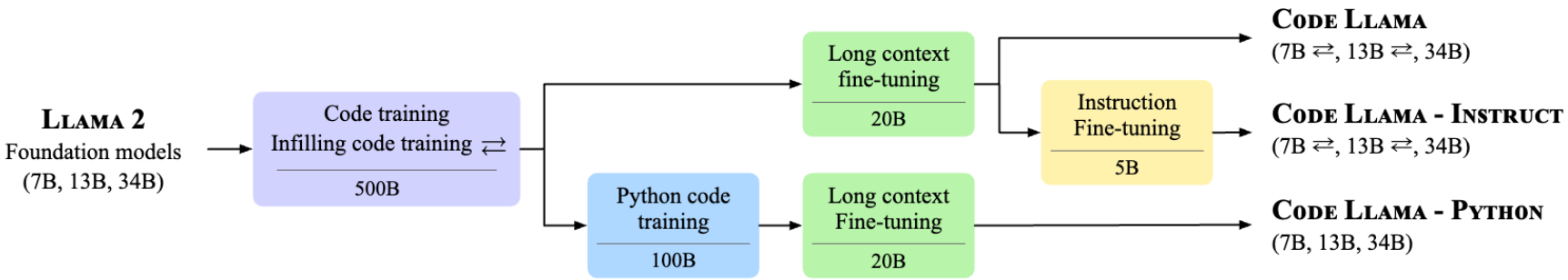
Input: x

Output:

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

Code Assistant: CodeLlama

- 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



Thank You!

Yu Meng
University of Virginia
yumeng5@virginia.edu