10-423/10-623 Generative AI

Machine Learning Department
School of Computer Science
Carnegie Mellon University

Code Generation
+
LLMs / VLMs as Autonomous Agents

Aran Nayebi & Matt Gormley
Lecture 23
Nov. 19, 2025

Reminders

- **Project Midway Report**
  - Due: Mon, Nov 24 at 11:59pm
- **HW623**
  - Only for students registered in 10-623 & 723
  - Due: Mon, Dec 1 at 11:59pm
- **Quiz 6**
  - Mon, Dec 1 in-class
  - Only for students registered in 10-723
  - Covers L21-L24

CODE GENERATION

How can you boost your productivity as a programmer?

Pair Programming

Coding with an LLM

Building Code Models

Languages for LLMs

- LLMs are trained on massive quantities of text from the internet (e.g. trillions of tokens)
- Some LLMs cover a variety of human languages, some focus primarily on English
- Most LLMs include **a wide variety of programming languages**

Programming Languages on GitHub

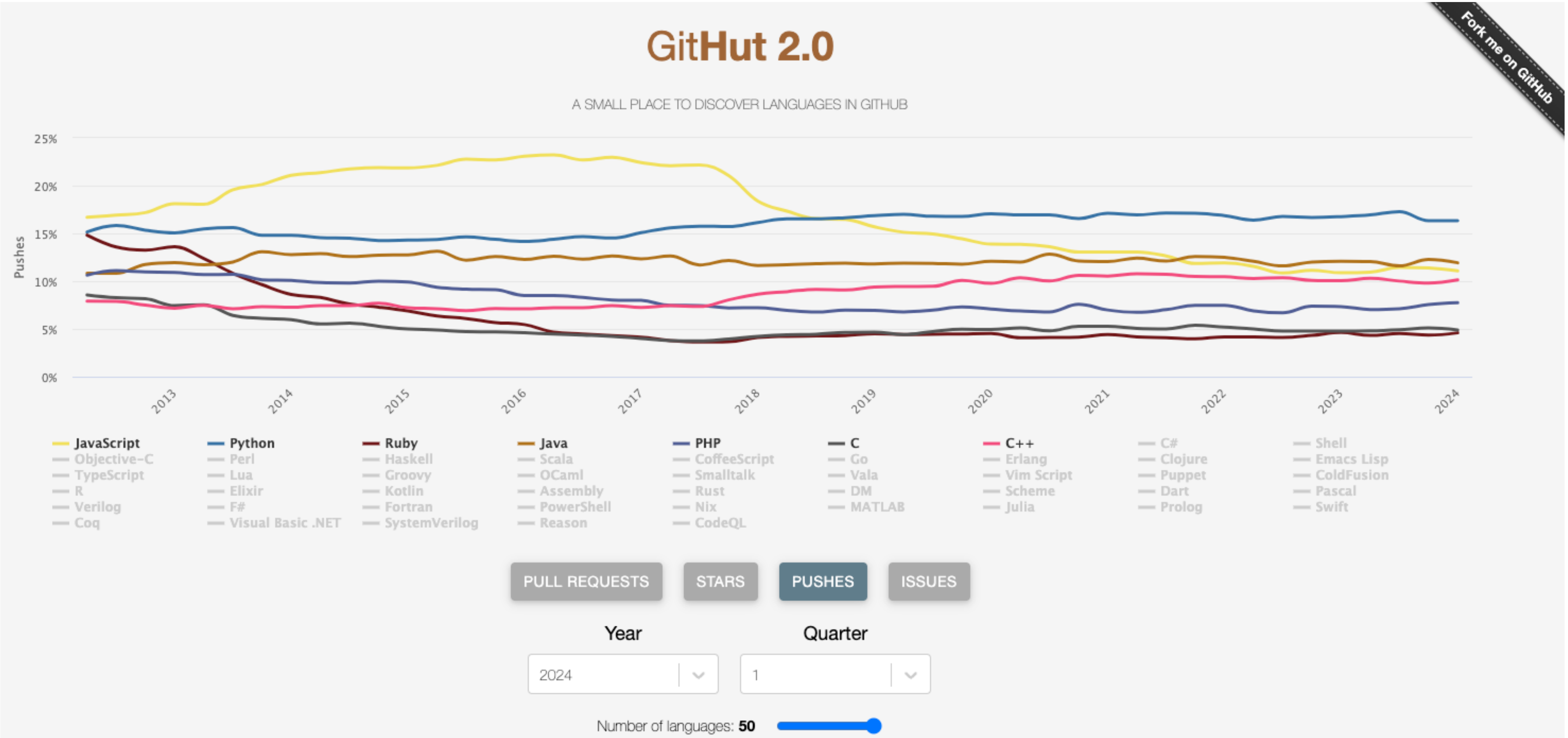


Figure from https://madnight.github.io/githut/#/pushes/2024/1

Figure from https://allenai.github.io/dolma/docs/assets/dolma-v0_1-20230819.pdf

Figure from https://arxiv.org/pdf/2211.15533

Example: Dolma Dataset

- The **Dolma** dataset is 3 trillion tokens of text
- It is sourced from a variety of existing datasets

Subset		Size		
Source	*Kind*	*Gzip files (GB)*	*Documents (millions)*	*Tokens (billions)*
Common Crawl 24 shards, 2020-05 to 2023-06	web	4,197	4,600	2,415
C4 [24] [8]	web	302	364	175
peS2o [27]	academic	150	38.8	57
The Stack [16]	code	675	236	430
Project Gutenberg	books	6.6	0.052	4.8
Wikipedia, Wikibooks *(en, simple)*	encyclopedic	5.8	6.1	3.6
Total		**5,334**	**5,245**	**3,084**

Table 1: Composition of Dolma.

- **The Stack** is 3TB of permissively licensed code intended for LLMs

Language	The Stack†	CodeParrot†	AlphaCode	CodeGen	PolyCoder†
Assembly	2.36	0.78			
Batchfile	1.00	0.7			
C	222.88	183.83		48.9	55
C#	128.37	36.83	38.4		21
C++	192.84	87.73	290.5	69.9	52
CMake	1.96	0.54			
CSS	145.33	22.67			
Dockerfile	1.95	0.71			
FORTRAN	3.10	1.62			
GO	118.37	19.28	19.8	21.4	15
Haskell	6.95	1.85			
HTML	746.33	118.12			
Java	271.43	107.7	113.8	120.3	41
JavaScript	486.20	87.82	88	24.7	22
Julia	3.09	0.29			
Lua	6.58	2.81	2.9		
Makefile	5.09	2.92			
Markdown	164.61	23.09			
Perl	5.50	4.7			
PHP	183.19	61.41	64		13
PowerShell	3.37	0.69			
Python	190.73	52.03	54.3	55.9 (217.3)	16
Ruby	23.82	10.95	11.6		4.1
Rust	40.35	2.68	2.8		3.5
Scala	14.87	3.87	4.1		1.8
Shell	8.69	3.01			
SQL	18.15	5.67			
TeX	4.65	2.15			
TypeScript	131.46	24.59	24.90		9.20
Visual Basic	2.73	1.91			
Total	3135.95	872.95	715.1	314.1	253.6

Table 1: The size of The Stack (in GB) compared to other source code datasets used for pre-training LLMs. † indicates the dataset is publicly released. The Stack is more than three times the size of CodeParrot, the next-largest released code dataset.

Applications for Code Models

Languages for LLMs

- LLMs are trained on massive quantities of text from the internet (e.g. trillions of tokens)
- Some LLMs cover a variety of human languages, some focus primarily on English
- Most LLMs include **a wide variety of programming languages**

Code Generation Examples

- Auto-complete for code IDEs (e.g. GitHub CoPilot)
- Write code given a text prompt
- Write code given a docstring (e.g. function-level or class-level comment)
- Read a large codebase and add a new feature
- Find bugs / fix bugs
- Write unit tests
- Translate code from one language to another
- Generate comments for existing code

Applications for Code Models

Type	I-O	Task	Definition
Understanding	C-K	Code Classification	Classify code snippets based on functionality, purpose, or attributes to aid in organization and analysis.
		Bug Detection	Detect and diagnose bugs or vulnerabilities in code to ensure functionality and security.
		Clone Detection	Identifying duplicate or similar code snippets in software to enhance maintainability, reduce redundancy, and check plagiarism.
		Exception Type Prediction	Predict different exception types in code to manage and handle exceptions effectively.
	C-C	Code-to-Code Retrieval	Retrieve relevant code snippets based on a given code query for reuse or analysis.
	NL-C	Code Search	Find relevant code snippets based on natural language queries to facilitate coding and development tasks.
Generation	C-C	Code Completion	Predict and suggest the next portion of code, given contextual information from the prefix (and suffix), while typing to enhance development speed and accuracy.
		Code Translation	Translate the code from one programming language to another while preserving functionality and logic.
		Code Repair	Identify and fix bugs in code by generating the correct version to improve functionality and reliability.
		Mutant Generation	Generate modified versions of code to test and evaluate the effectiveness of testing strategies.
		Test Generation	Generate test cases to validate code functionality, performance, and robustness.
	C-NL	Code Summarization	Generate concise textual descriptions or explanations of code to enhance understanding and documentation.
	NL-C	Code Generation	Generate source code from natural language descriptions to streamline development and reduce manual coding efforts.

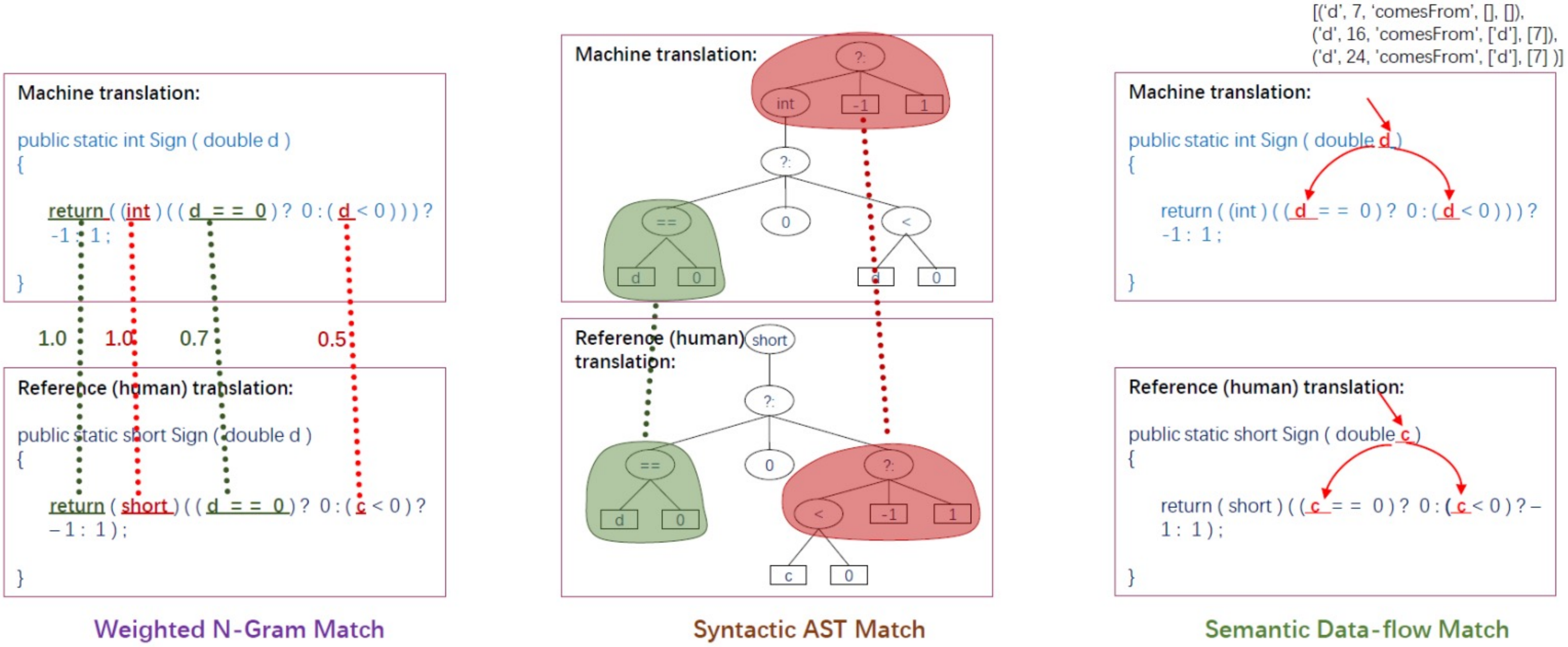
Figure from https://arxiv.org/pdf/2406.00515

EVALUATING CODE GENERATION

How to evaluate code?

- Metrics
  - BLEU (metric used in machine translation for testing n-gram overlap with a known reference)
  - CodeBLEU (a mixture of various syntactic and semantic metrics)
  - Functional correctness (checks how many unit tests pass)
  - NOTE: functional correctness has become the dominant metric for evaluation
- Benchmarks
  - HumanEval
  - MBPP
  - (many more!)

CodeBLEU



CodeBLEU $= \alpha \cdot$ N – Gram Match (BLEU) $+ \beta \cdot$ Weighted N-Gram Match $+ \gamma \cdot$ Syntactic AST Match $+ \delta \cdot$ Semantic Data-flow Match

Figure from https://arxiv.org/pdf/2009.10297

HumanEval Benchmark

- Introduced alongside Codex model
- Measures functional correctness of code (i.e. how many unit tests pass)
- pass@k metric = % of k code samples that pass *all* the unit tests (actual implementation uses more samples to reduce variance)
- 164 handwritten problems

```python
def incr_list(l: list):
    """Return list with elements incremented by 1.
    >>> incr_list([1, 2, 3])
    [2, 3, 4]
    >>> incr_list([5, 3, 5, 2, 3, 3, 9, 0, 123])
    [6, 4, 6, 3, 4, 4, 10, 1, 124]
    """
    return [i + 1 for i in l]
```

```python
def solution(lst):
    """Given a non-empty list of integers, return the sum of all of the odd elements
    that are in even positions.

    Examples
    solution([5, 8, 7, 1]) =⇒12
    solution([3, 3, 3, 3, 3]) =⇒9
    solution([30, 13, 24, 321]) =⇒0
    """
    return sum(lst[i] for i in range(0,len(lst)) if i % 2 == 0 and lst[i] % 2 == 1)
```

```python
def encode_cyclic(s: str):
    """
    returns encoded string by cycling groups of three characters.
    """
    # split string to groups. Each of length 3.
    groups = [s[(3 * i):min((3 * i + 3), len(s))] for i in range((len(s) + 2) // 3)]
    # cycle elements in each group. Unless group has fewer elements than 3.
    groups = [(group[1:] + group[0]) if len(group) == 3 else group for group in groups]
    return "".join(groups)

def decode_cyclic(s: str):
    """
    takes as input string encoded with encode_cyclic function. Returns decoded string.
    """
    # split string to groups. Each of length 3.
    groups = [s[(3 * i):min((3 * i + 3), len(s))] for i in range((len(s) + 2) // 3)]
    # cycle elements in each group.
    groups = [(group[-1] + group[:-1]) if len(group) == 3 else group for group in groups]
    return "".join(groups)
```

Figure from http://arxiv.org/abs/2107.03374

HumanEval Benchmark

- Introduced alongside Codex model
- Measures functional correctness of code (i.e. how many unit tests pass)
- pass@k metric = % of k code samples that pass *all* the unit tests (actual implementation uses more samples to reduce variance)
- 164 handwritten problems

- Results clearly indicate that BLEU is not a good surrogate

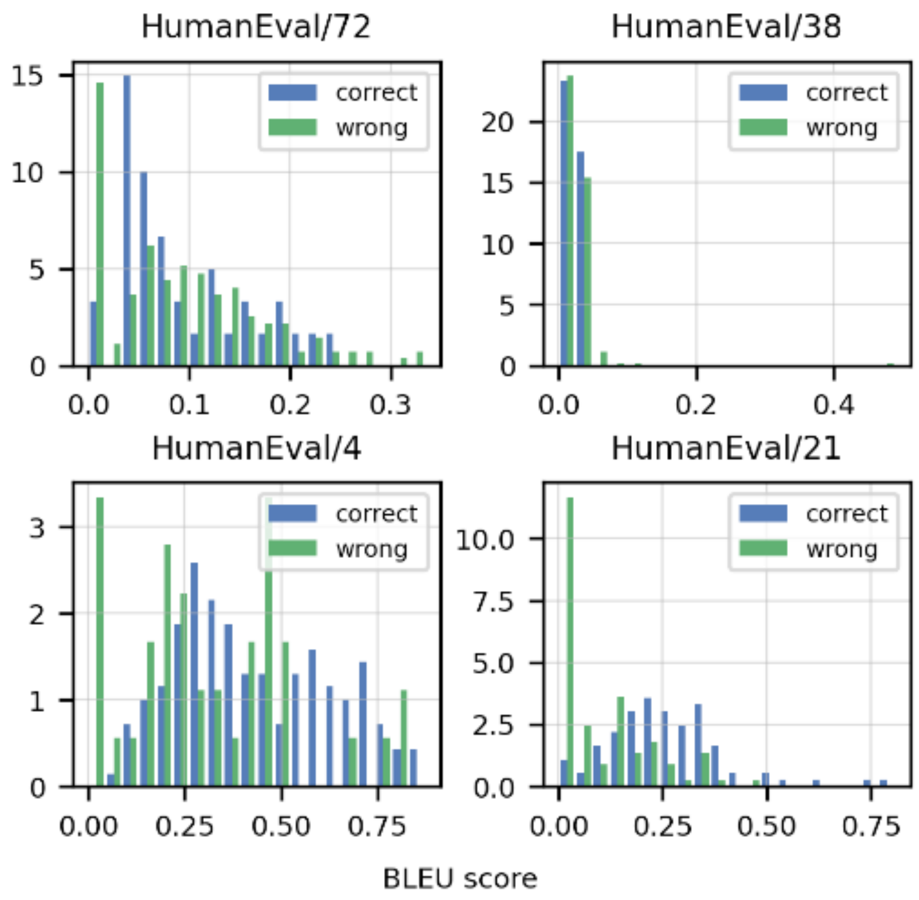


Figure 8. BLEU score probability densities for correct (blue) and wrong (green) solutions from Codex-12B for 4 random tasks from HumanEval. Note that the distributions are not cleanly separable, suggesting that optimizing for BLEU score is not equivalent to optimizing for functional correctness.

Figure from http://arxiv.org/abs/2107.03374

MBPP

- Mostly Basic Python Programs (MBPP) are intended to be problems solvable by a novice programmer
- 974 problems constructed by crowd-sourcing
- Consists of problem statement, 3 tests, and a self-contained solution

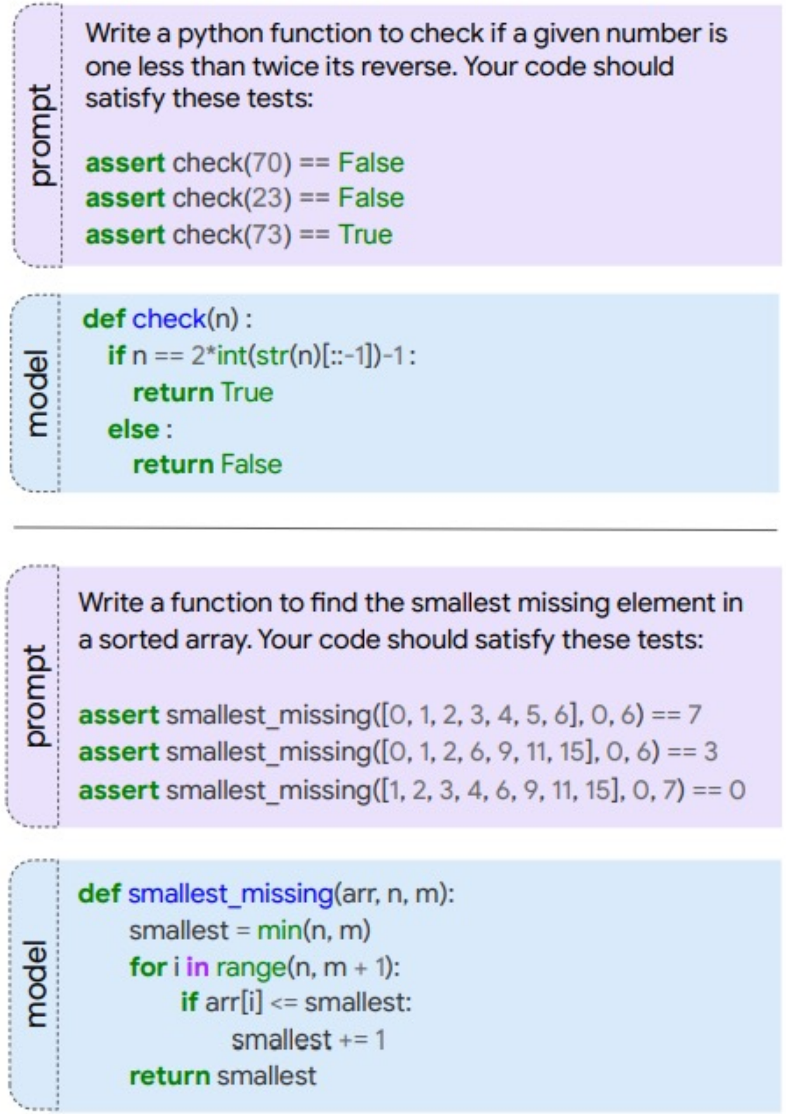


Figure from https://arxiv.org/pdf/2108.07732

CODE MODELS

Code Generation with LLMs

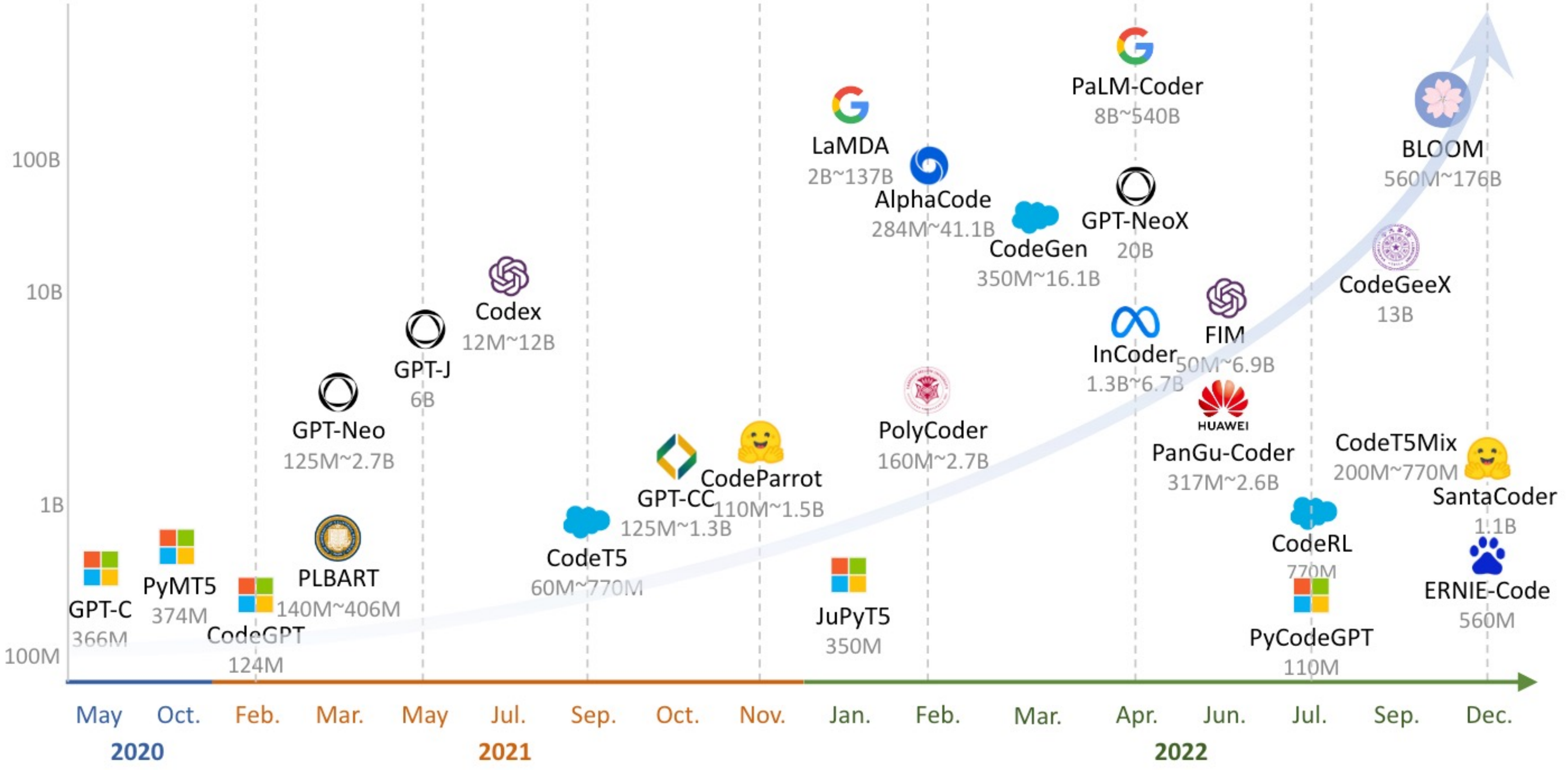


Figure from https://aclanthology.org/2023.acl-long.411.pdf

Approaches to Code Generation

Here we consider a few representative examples of code models:

- CodeBERT
- Codex
- CodeT5
- InCoder / FIM
- StarCoder
- LongCoder

CodeBERT

- One of the early successes in this space, CodeBERT has 125M model parameters (same architecture as RoBERTa)
- Two pre-training objectives:
  – Masked language modeling (MLM)
  – Replaced token detection (RTD)

- Example application:
  – natural language code retrieval

MODEL	RUBY	JAVASCRIPT	GO	PYTHON
NBow	0.4285	0.4607	0.6409	0.5809
CNN	0.2450	0.3523	0.6274	0.5708
BiRNN	0.0835	0.1530	0.4524	0.3213
selfAtt	0.3651	0.4506	0.6809	0.6922
RoBERTa	0.6245	0.6060	0.8204	0.8087
PT w/ Code Only (init=s)	0.5712	0.5557	0.7929	0.7855
PT w/ Code Only (init=R)	0.6612	0.6402	0.8191	0.8438
CodeBERT (MLM, init=s)	0.5695	0.6029	0.8304	0.8261
CodeBERT (MLM, init=R)	0.6898	0.6997	0.8383	0.8647
CodeBERT (RTD, init=R)	0.6414	0.6512	0.8285	0.8263
CodeBERT (MLM+RTD, init=R)	**0.6926**	**0.7059**	**0.8400**	**0.8685**

Figure from http://arxiv.org/abs/2002.08155

Codex

- The original model behind GitHub Copilot
- GPT-3 model with 12B parameters fine-tuned on 159 GB of Python code
- Notably: using a pre-trained GPT-3 does not improve performance, but does improve convergence time

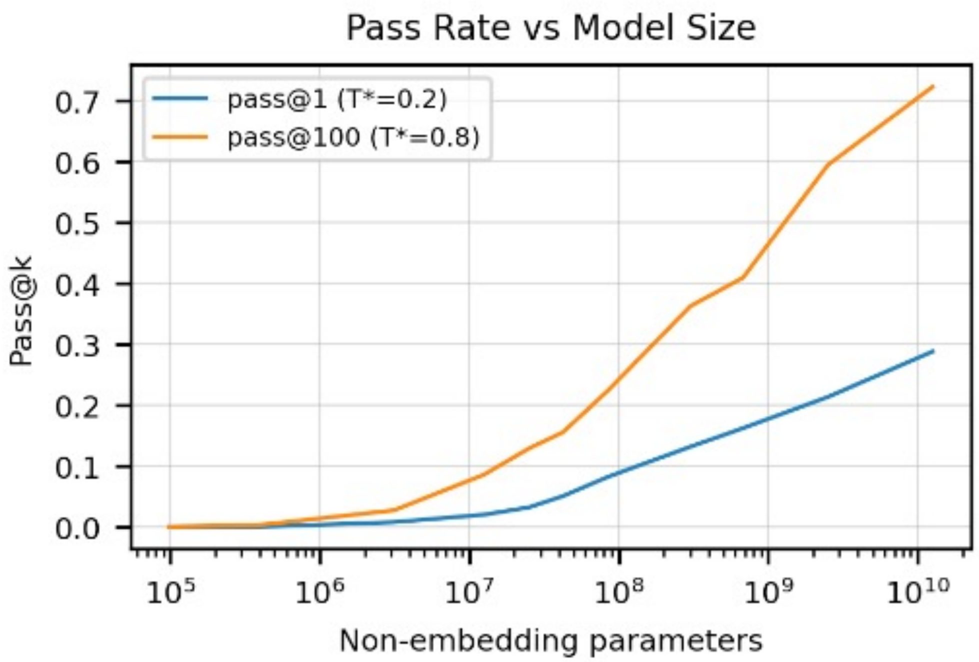
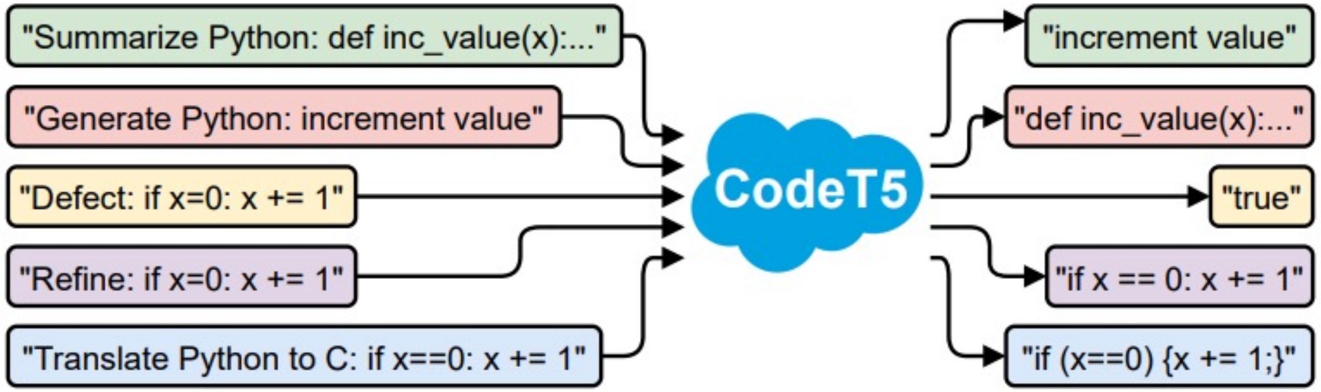


Figure from http://arxiv.org/abs/2107.03374

CodeT5

- CodeT5 is based on the T5 encoder-decoder Transformer architecture
- Like T5, CodeT5 brings together a number of different tasks



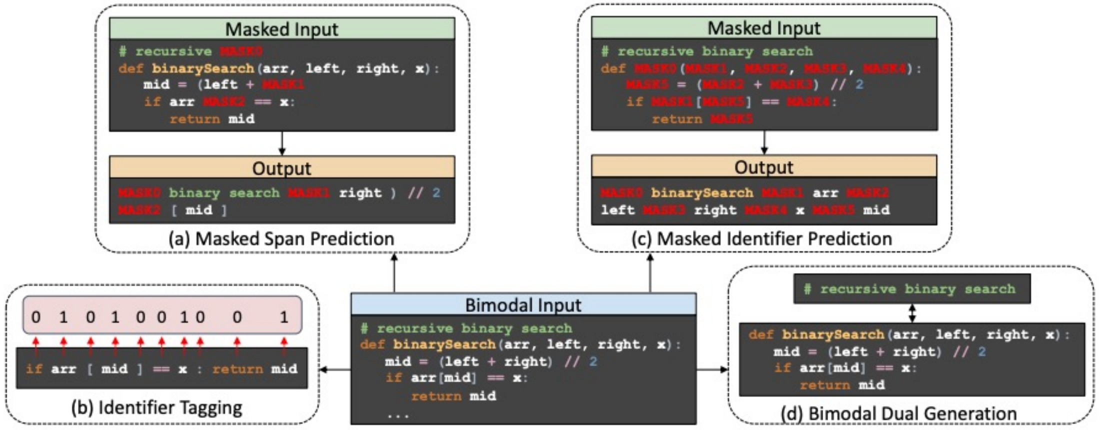


Figure 2: Pre-training tasks of CodeT5. We first alternately train span prediction, identifier prediction, and identifier tagging on both unimodal and bimodal data, and then leverage the bimodal data for dual generation training.

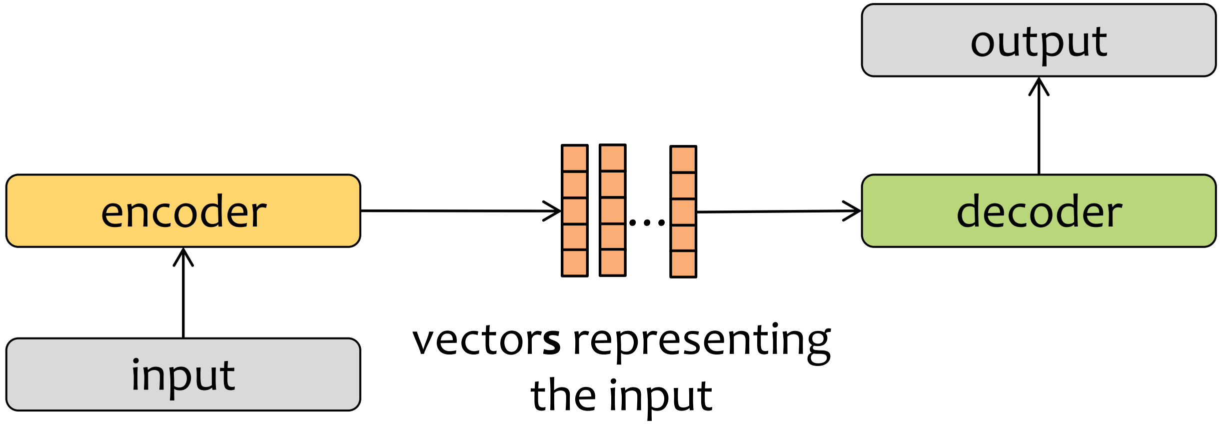


Figure from https://arxiv.org/pdf/2109.00859

Question how do we use an **causally-masked LM** to fill in code in the **middle** of code file?

InCoder / FIM

InCoder (April 2022)

- Place a mask token where you want to fill in the code

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>
```

FIM (July 2022)

- Goal is to train a model that fills in the middle
- Divide the code snippet into (prefix, middle, suffix)
- Then train with examples of the form:

  <PRE> prefix <SUF> suffix <MID> middle

- And predict using examples of the form:

  <PRE> prefix <SUF> suffix <MID>

Figure from https://arxiv.org/pdf/2204.05999

FIM paper https://arxiv.org/pdf/2207.14255

StarCoder

StarCoder

- (Was) one of the best open source Code Models
- Used FIM for pre-training a 15.5B parameter model on 1 trillion tokens of text

Hyperparameter	SantaCoder	StarCoder
Hidden size	2048	6144
Intermediate size	8192	24576
Max. position embeddings	2048	8192
Num. of attention heads	16	48
Num. of hidden layers	24	40
Attention	Multi-query	Multi-query
Num. of parameters	$\approx$ 1.1B	$\approx$15.5B

Model	Size	HumanEval	MBPP
Open-access			
LLaMA	7B	10.5	17.7
LLaMA	13B	15.8	22.0
SantaCoder	1.1B	18.0	35.0
CodeGen-Multi	16B	18.3	20.9
LLaMA	33B	21.7	30.2
CodeGeeX	13B	22.9	24.4
LLaMA-65B	65B	23.7	37.7
CodeGen-Mono	16B	29.3	35.3
StarCoderBase	15.5B	30.4	49.0
StarCoder	15.5B	33.6	52.7
Closed-access			
LaMDA	137B	14.0	14.8
PaLM	540B	26.2	36.8
code-cushman-001	12B	33.5	45.9
code-davinci-002	175B	45.9	60.3

LongCoder

- LongCoder aims to address the problem of working with large codebases
- Employs sparse attention to handle long input sequences

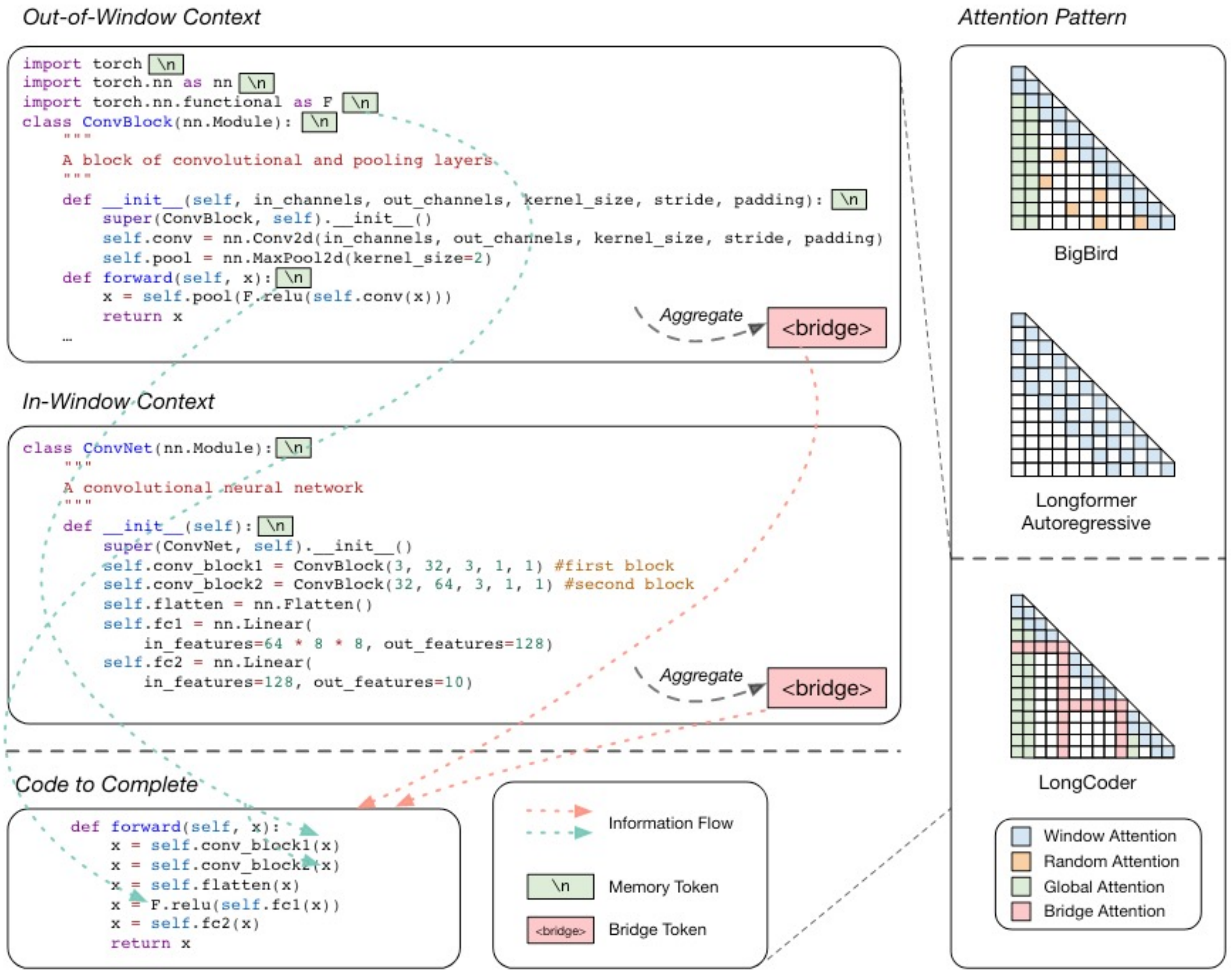


Figure 1. ***(Left)*** An example of how LongCoder facilitates completion with longer context. The memory tokens save potentially useful information (including package imports, class and function definitions) for global access despite whether they are within the sliding window. The bridge tokens aggregate local information by attending to a fixed length of tokens. The information flow within the window is omitted for clarity. ***(Right)*** Attention patterns used in BigBird (Zaheer et al., 2020), Longformer (Beltagy et al., 2020) and LongCoder. Best viewed in color.

CODE MODEL-SPECIFIC TECHNIQUES

Iterative Self-Refinement

- Normally self-correction with an LLM (e.g. for reasoning problems) does not work
- However, when performing self-correction on a code model, we may also have access to unit test output
- This output from unit tests can lead to great success in iterative self-refinement at test time

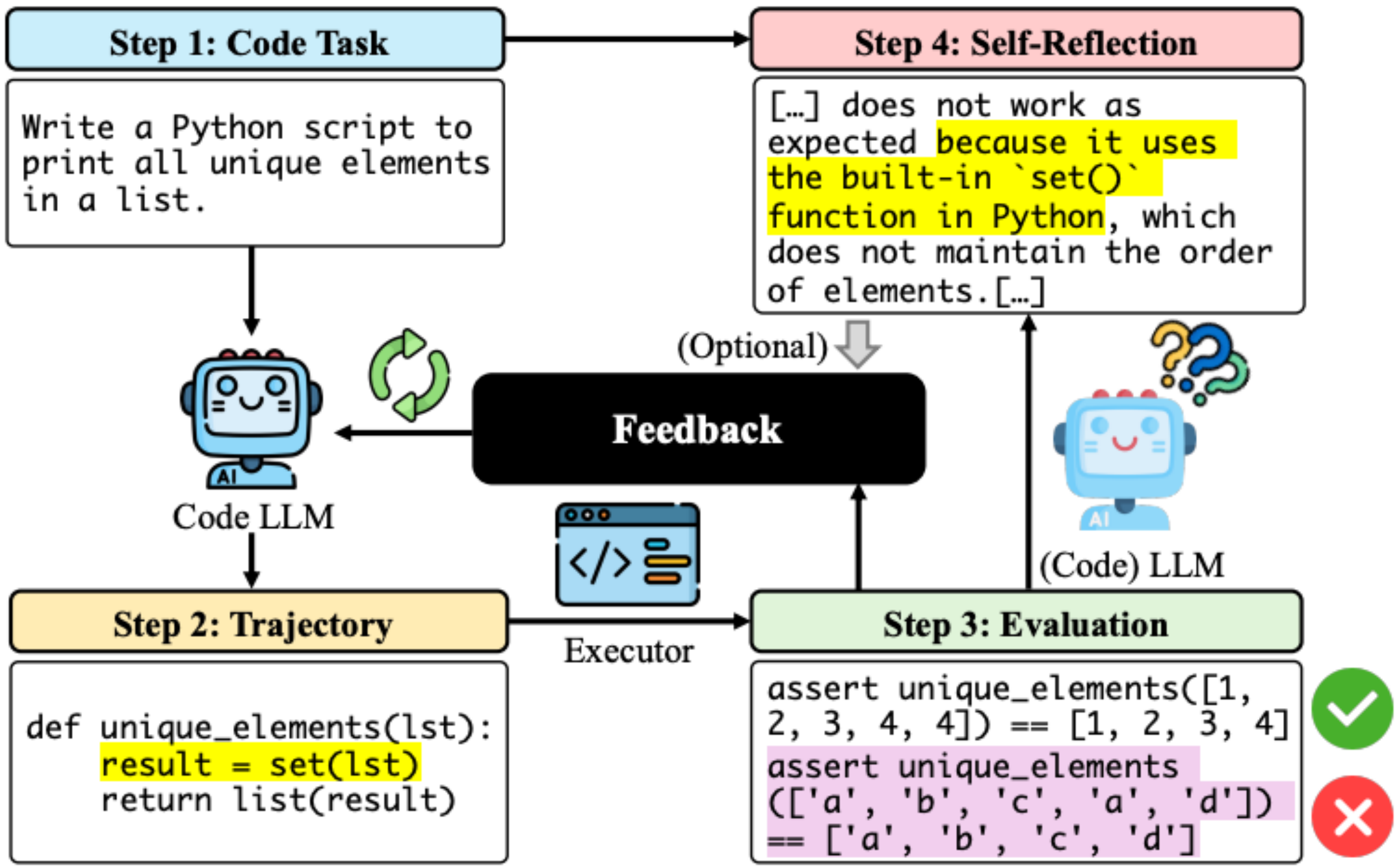
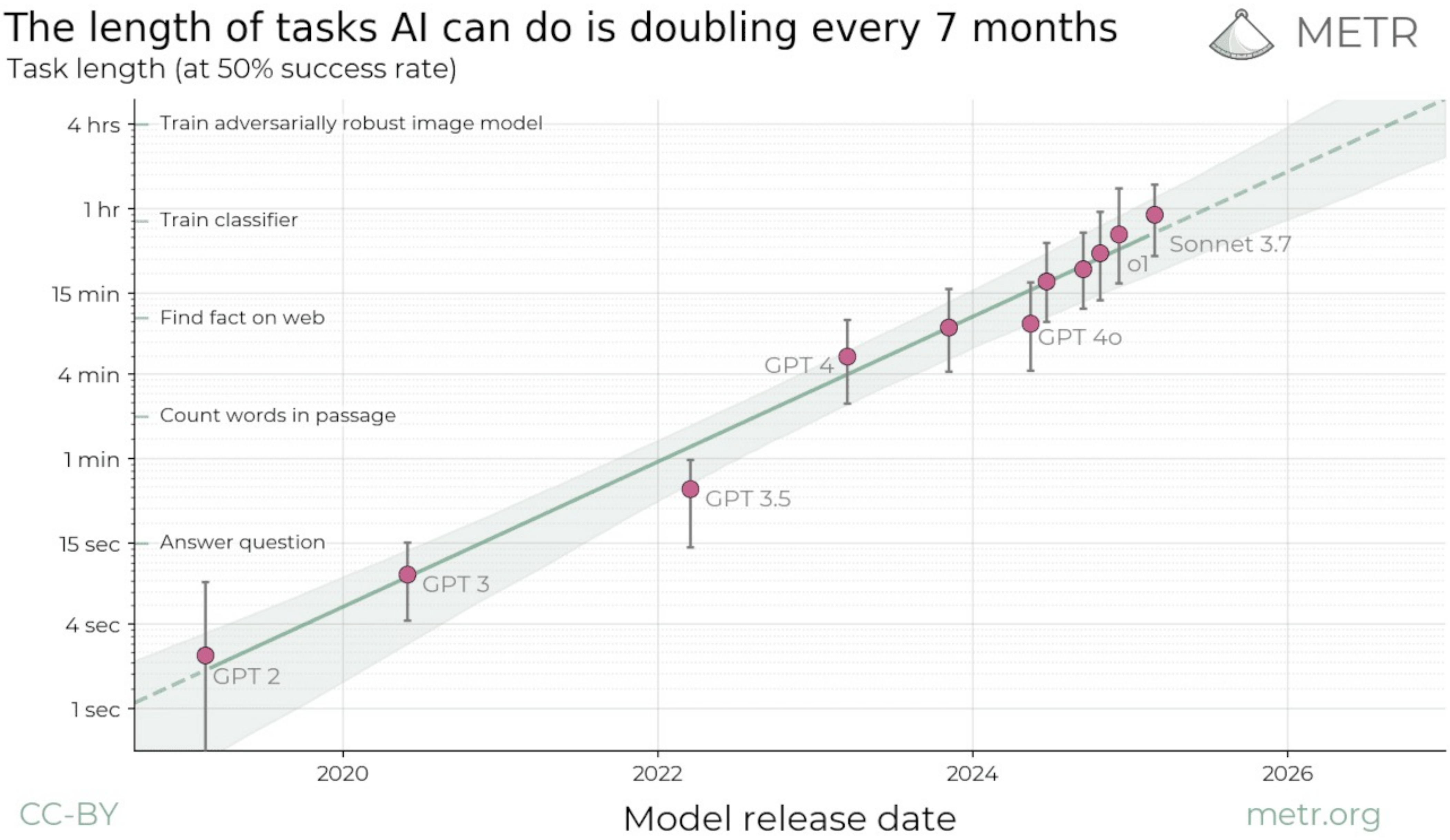


Figure from https://arxiv.org/pdf/2406.00515

Do code assistants make programmers more efficient?



AUTONOMOUS AGENTS

LLM-based Autonomous Agents

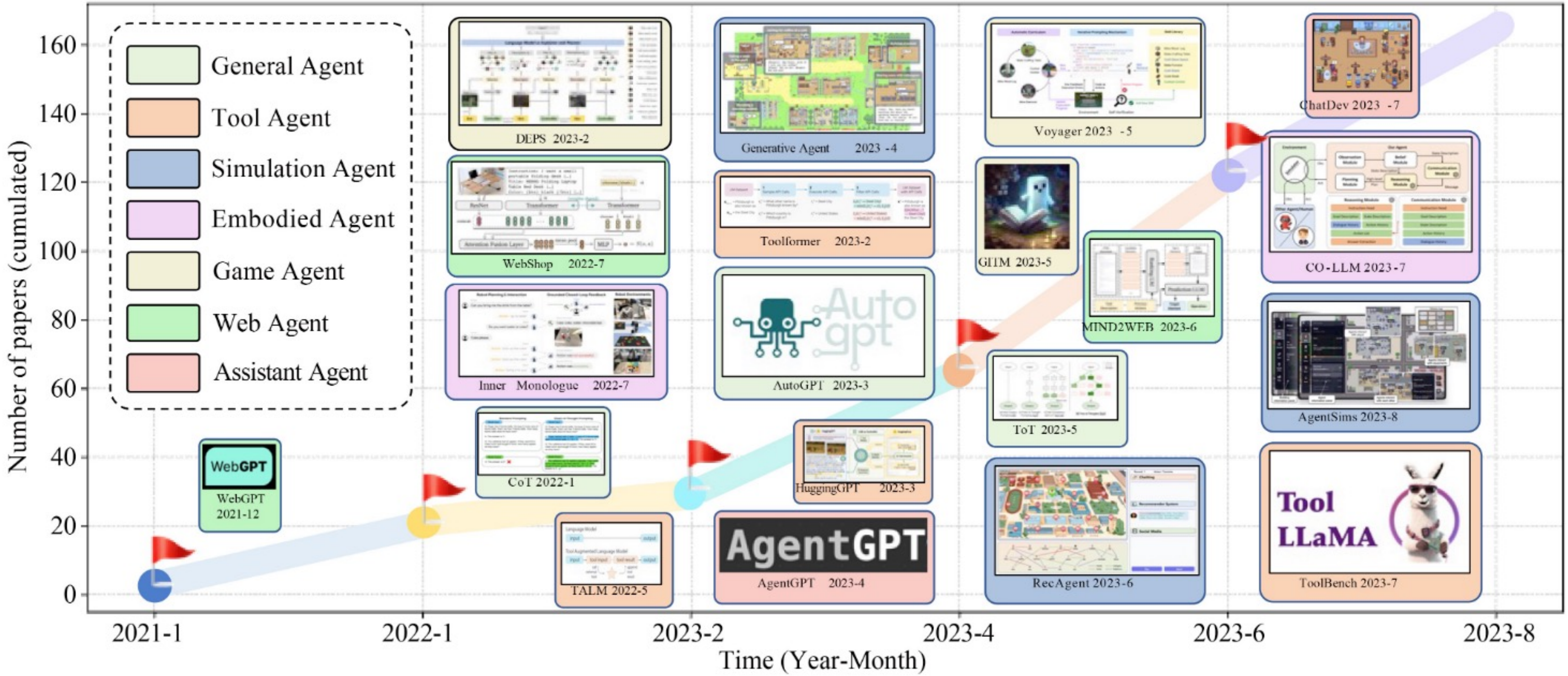


Fig. 1 Illustration of the growth trend in the field of LLM-based autonomous agents. We present the cumulative number of papers published from January 2021 to August 2023. We assign different colors to represent various agent categories. For example, a game agent aims to simulate a game-player, while a tool agent mainly focuses on tool using. For each time period, we provide a curated list of studies with diverse agent categories

Large Language Models as Agents

- **Mind2Web** enables interaction with HTML generated GUIs through direct access to the HTML
- The system consists of two models (1) Ranking LM and (2) Prediction LLM

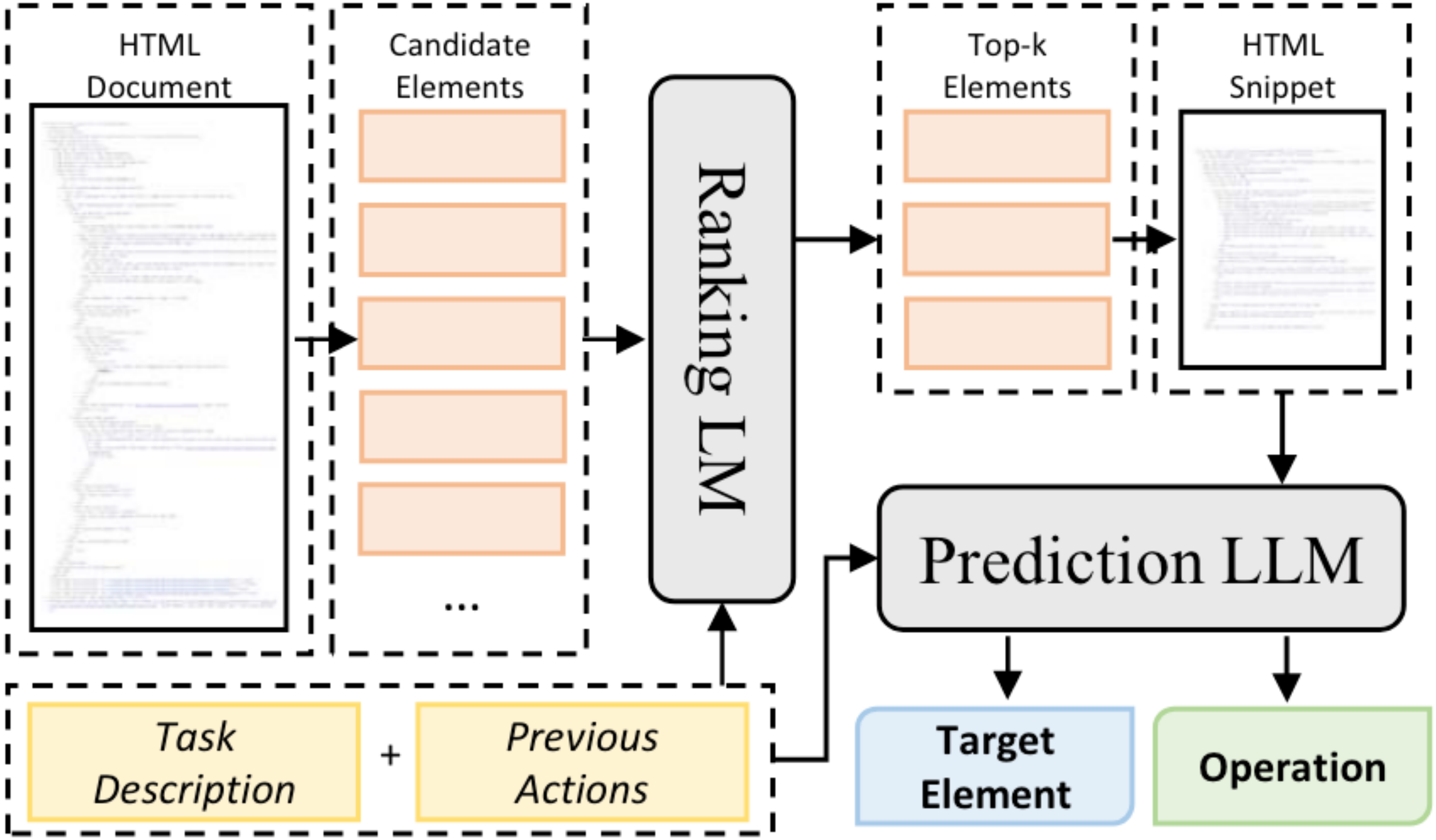


Figure from http://arxiv.org/abs/2306.06070

Large Language Models as Agents

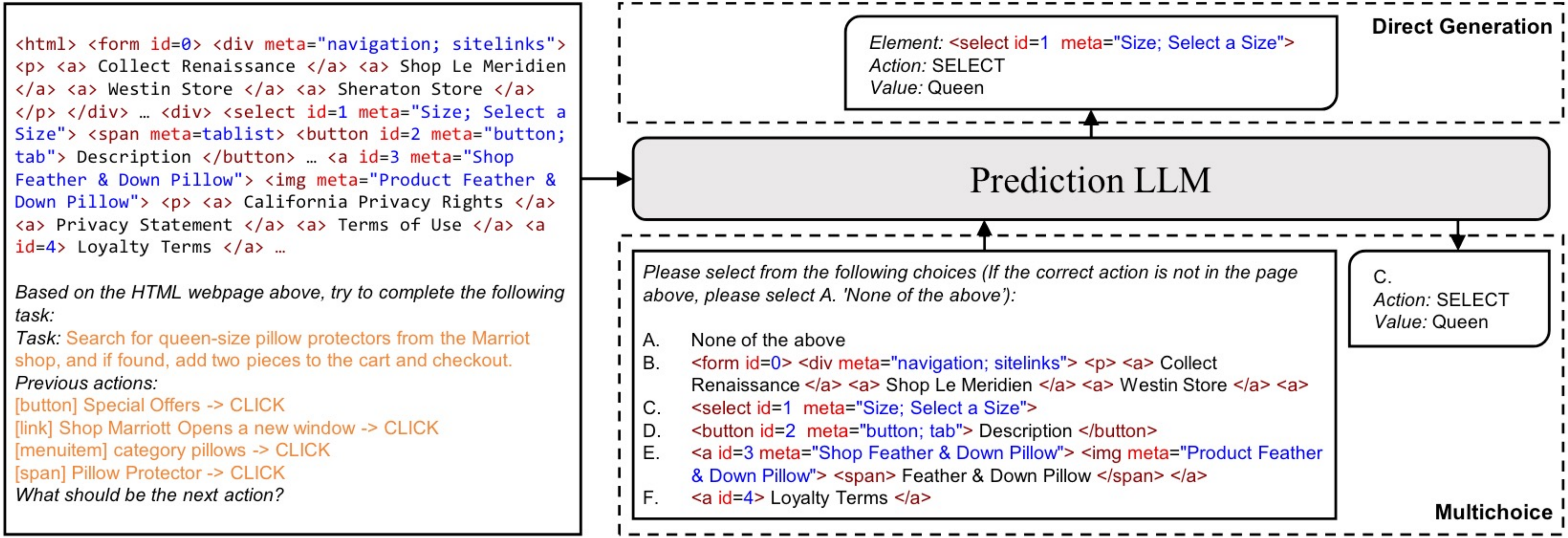


Figure from http://arxiv.org/abs/2306.06070

Visual Language Models as Agents

- A typical VLM can be used to develop an agent for autonomous interaction with a graphical user interface (GUI)
- Typically, there are two key components:
  - Visual grounding model: this part identifies the next action to take, e.g. where to click/type/interact with the GUI based on the prompt and the current screenshot
  - GUI agent model: this guides the overall interaction to achieve the end goal

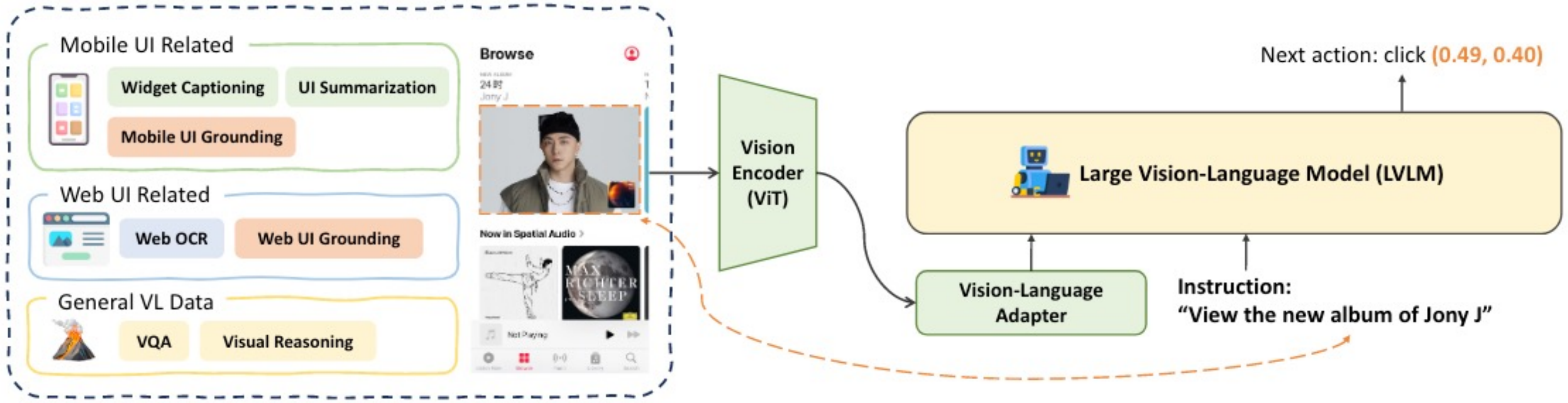


Figure from http://arxiv.org/abs/2401.10935

Gemini 3 Pro (released yesterday!)

https://blog.google/products/gemini/gemini-3/#build-anything

Benchmark	Description		Gemini 3 Pro	Gemini 2.5 Pro	Claude Sonnet 4.5	GPT-5.1
Humanity's Last Exam	Academic reasoning	No tools	**37.5%**	21.6%	13.7%	26.5%
		With search and code execution	**45.8%**	—	—	—
ARC-AGI-2	Visual reasoning puzzles	ARC Prize Verified	**31.1%**	4.9%	13.6%	17.6%
GPQA Diamond	Scientific knowledge	No tools	**91.9%**	86.4%	83.4%	88.1%
AIME 2025	Mathematics	No tools	**95.0%**	88.0%	87.0%	94.0%
		With code execution	**100%**	—	**100%**	—
MathArena Apex	Challenging Math Contest problems		**23.4%**	0.5%	1.6%	1.0%
MMMU-Pro	Multimodal understanding and reasoning		**81.0%**	68.0%	68.0%	76.0%
ScreenSpot-Pro	Screen understanding		**72.7%**	11.4%	36.2%	3.5%
CharXiv Reasoning	Information synthesis from complex charts		**81.4%**	69.6%	68.5%	69.5%
OmniDocBench 1.5	OCR	Overall Edit Distance, lower is better	**0.115**	0.145	0.145	0.147
Video-MMMU	Knowledge acquisition from videos		**87.6%**	83.6%	77.8%	80.4%
LiveCodeBench Pro	Competitive coding problems from Codeforces, ICPC, and IOI	Elo Rating, higher is better	**2,439**	1,775	1,418	2,243
Terminal-Bench 2.0	Agentic terminal coding	Terminus-2 agent	**54.2%**	32.6%	42.8%	47.6%
SWE-Bench Verified	Agentic coding	Single attempt	76.2%	59.6%	**77.2%**	76.3%
τ2-bench	Agentic tool use		**85.4%**	54.9%	84.7%	80.2%
Vending-Bench 2	Long-horizon agentic tasks	Net worth (mean), higher is better	**$5,478.16**	$573.64	$3,838.74	$1,473.43
FACTS Benchmark Suite	Held out internal grounding, parametric, MM, and search retrieval benchmarks		**70.5%**	63.4%	50.4%	50.8%
SimpleQA Verified	Parametric knowledge		**72.1%**	54.5%	29.3%	34.9%
MMMLU	Multilingual Q&A		**91.8%**	89.5%	89.1%	91.0%
Global PIQA	Commonsense reasoning across 100 Languages and Cultures		**93.4%**	91.5%	90.1%	90.9%
MRCR v2 (8-needle)	Long context performance	128k (average)	**77.0%**	58.0%	47.1%	61.6%
		1M (pointwise)	**26.3%**	16.4%	not supported	not supported

For details on our evaluation methodology please see deepmind.google/models/evals-methodology/gemini-3-pro

Set-of-Mark Prompting

- **Key idea:** label the semantically meaningful components of each image with numbers
- The numbers are directly embedded into the image by changing the pixels
- The prompt can refer to the numbers or not

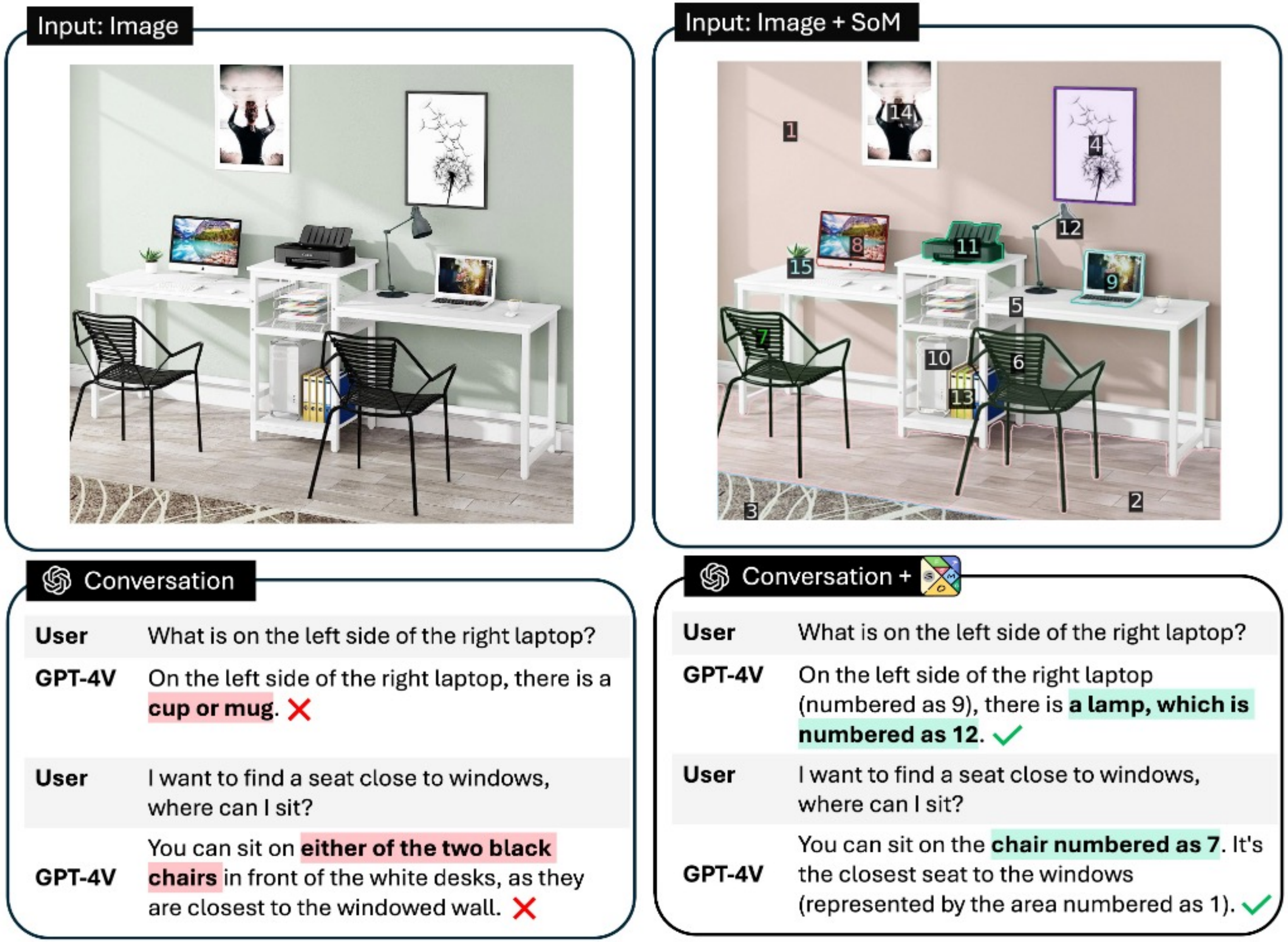


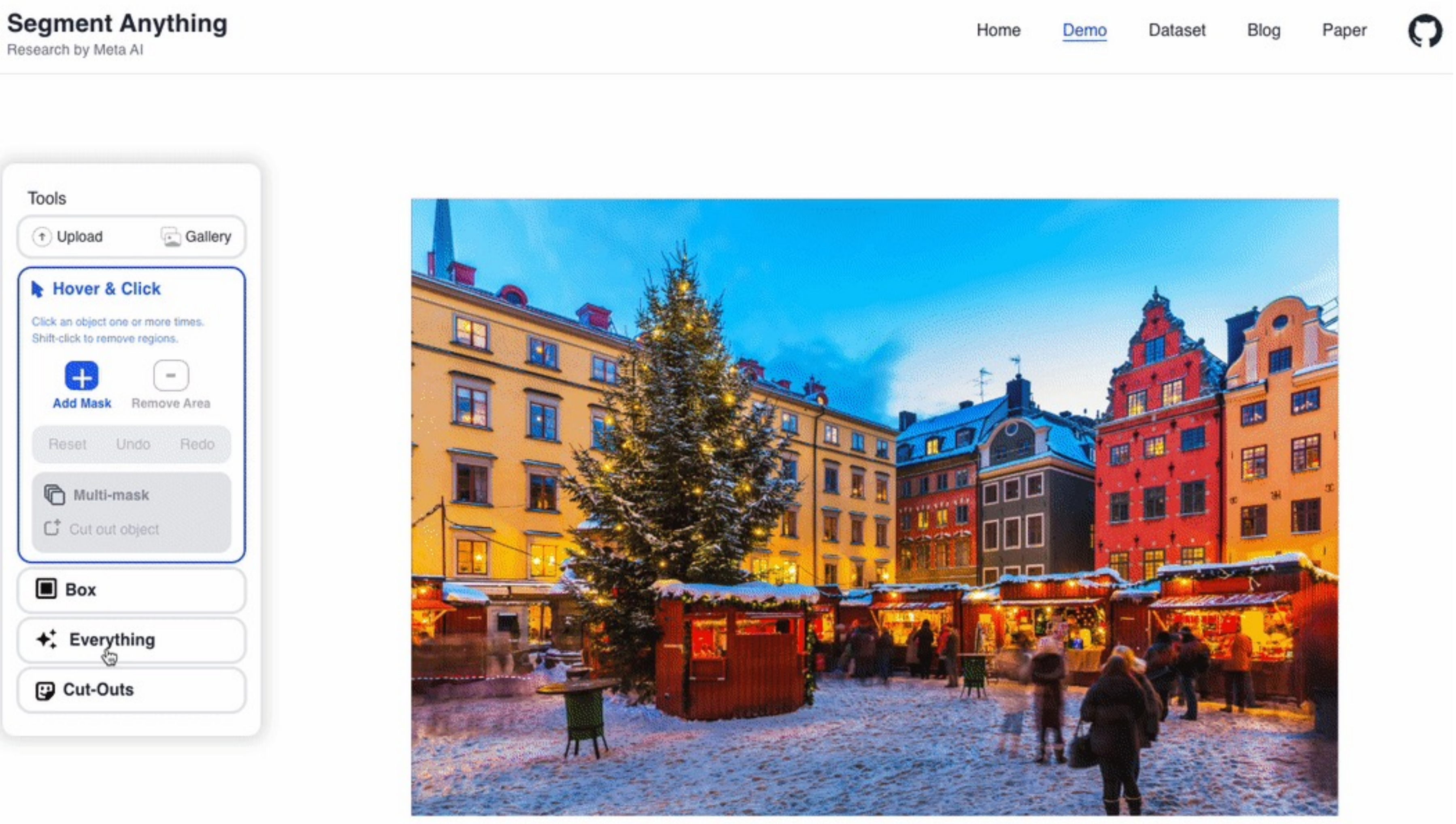
Figure 1: Comparisons of GPT-4V prompting techniques: (left) standard prompting and (right) the proposed *Set-of-Mark Prompting*. Simply overlaying ids on image regions unleashes visual grounding and corrects answers for GPT-4V. *Note that no marks are leaked to user text inputs.*.

Figure from http://arxiv.org/abs/2310.11441

Set-of-Mark Prompting

- **Key idea:** label the semantically meaningful components of each image with numbers
- The numbers are directly embedded into the image by changing the pixels
- The prompt can refer to the numbers or not

- **Question:** How do we identify the regions to label?
- **Answer:** Use an off-the-shelf segmentation model!



https://segment-anything.com/demo

Set-of-Mark Prompting

- **Key idea:** label the semantically meaningful components of each image with numbers
- The numbers are directly embedded into the image by changing the pixels
- The prompt can refer to the numbers or not

- **Question:** Does set-of-mark prompting work with an VLM?
- **Answer:** Not necessarily. The paper found that GPT-4V could interpret and ground itself in the marks. However, other models including Llava-1.5 and MiniGPT-v2 could not.

Figure from http://arxiv.org/abs/2310.11441

SeeClick

- Key Idea: directly train an off-the-shelf VLM (Qwen-VL) to perform individual actions by looking only at the screenshot

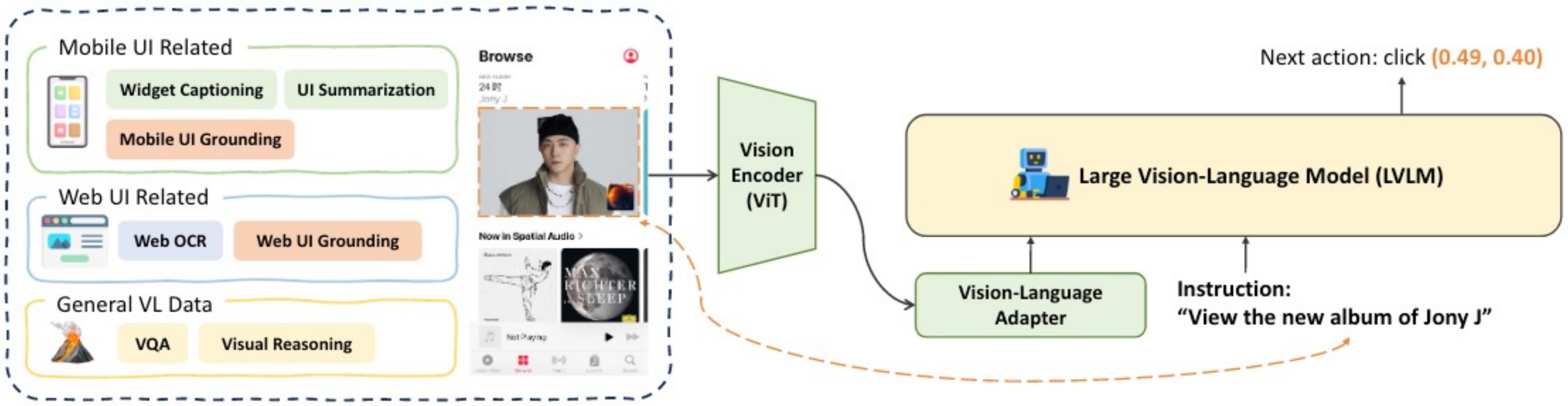


Figure from http://arxiv.org/abs/2401.10935

SeeClick

- Key Idea: directly train an off-the-shelf VLM (Qwen-VL) to perform individual actions by looking only at the screenshot

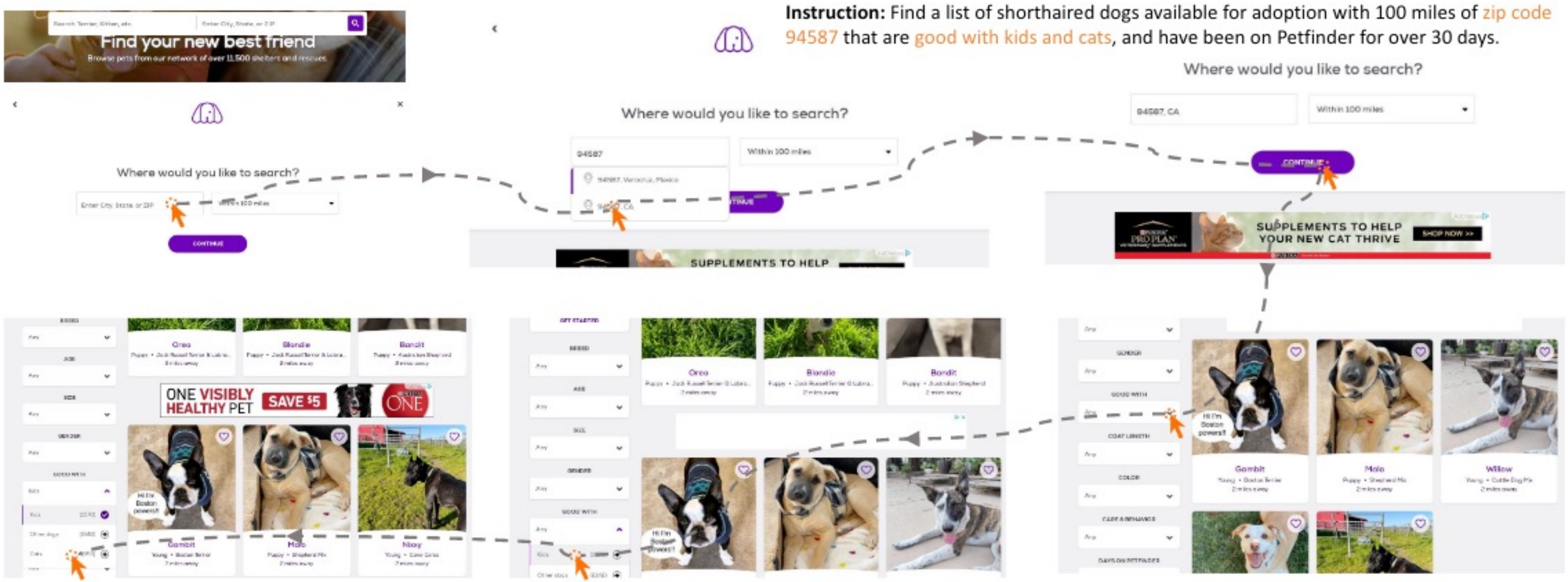
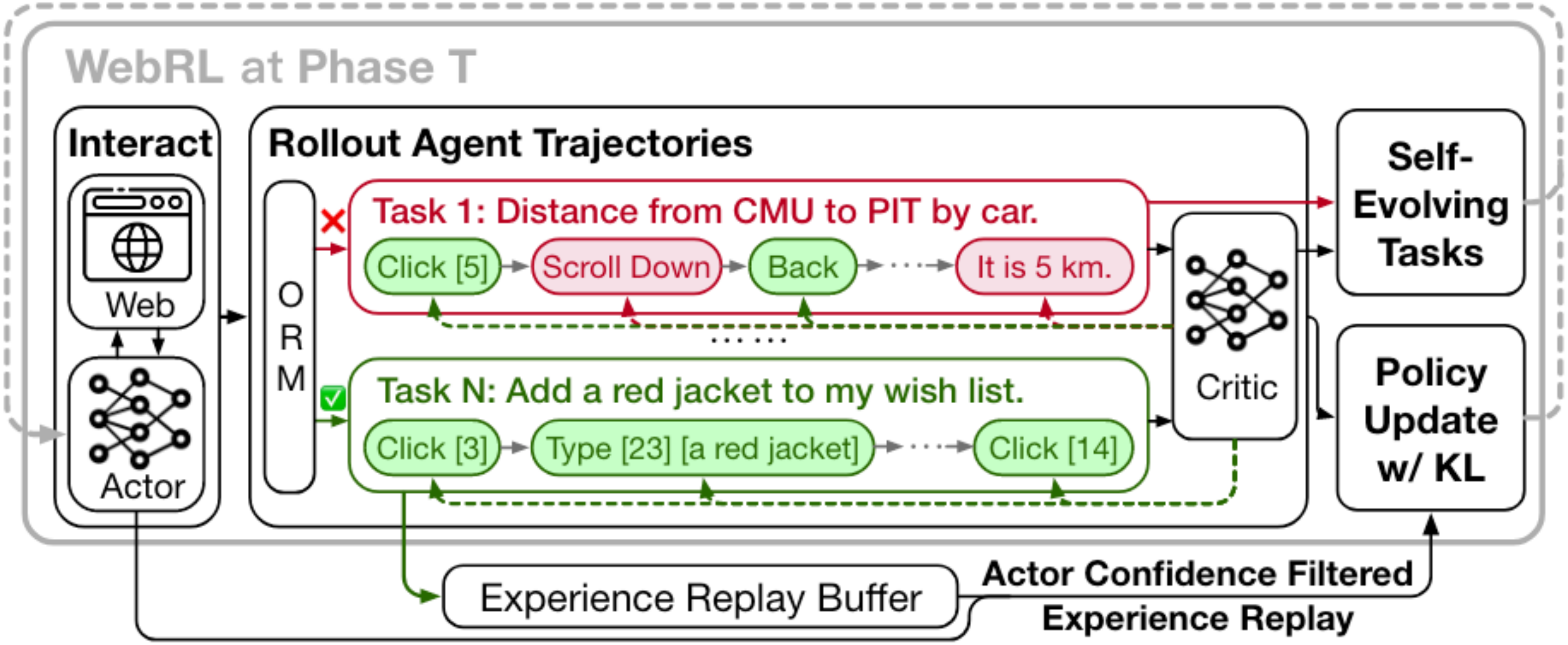


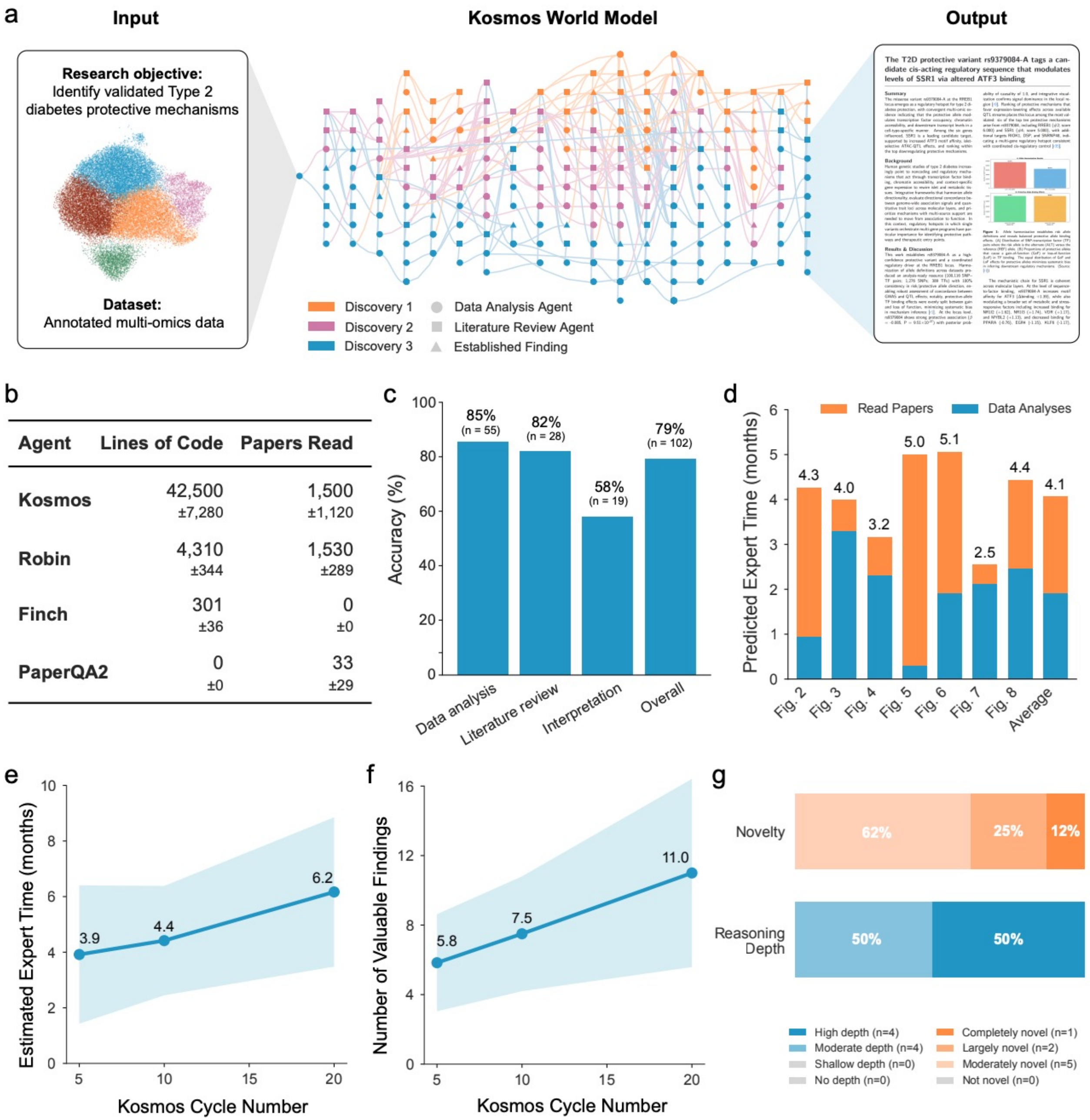
Figure from http://arxiv.org/abs/2401.10935

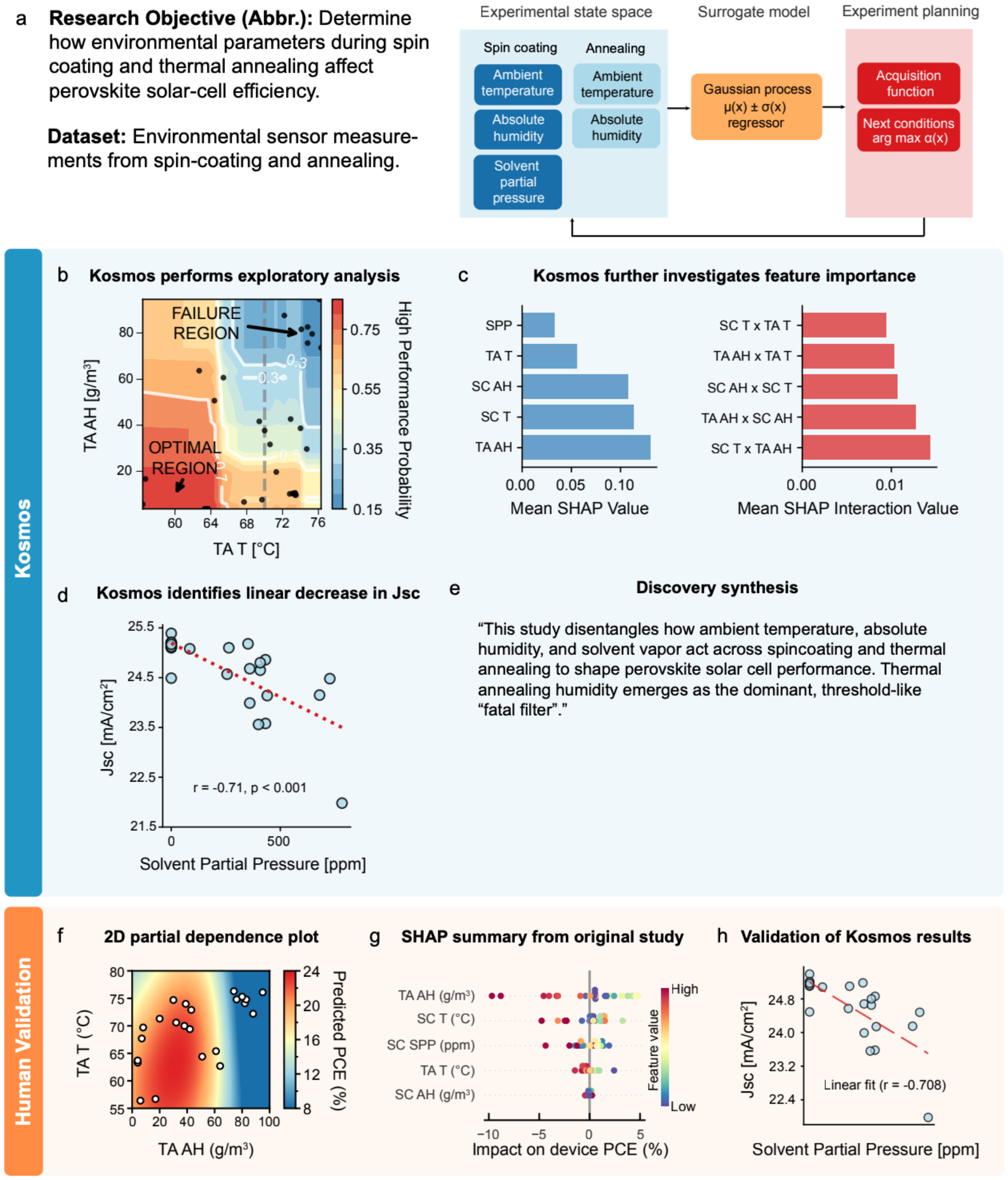
Using Reinforcement Learning to Train Agents

- Problem: there are actually many ways to complete a task, so how do we train a model without imitating a human's attempt at completing the task?
- Solution: use RL, where the positive reward comes at the end if the task was completed successfully



Agents as Scientists





Agents as Animal Brain Models

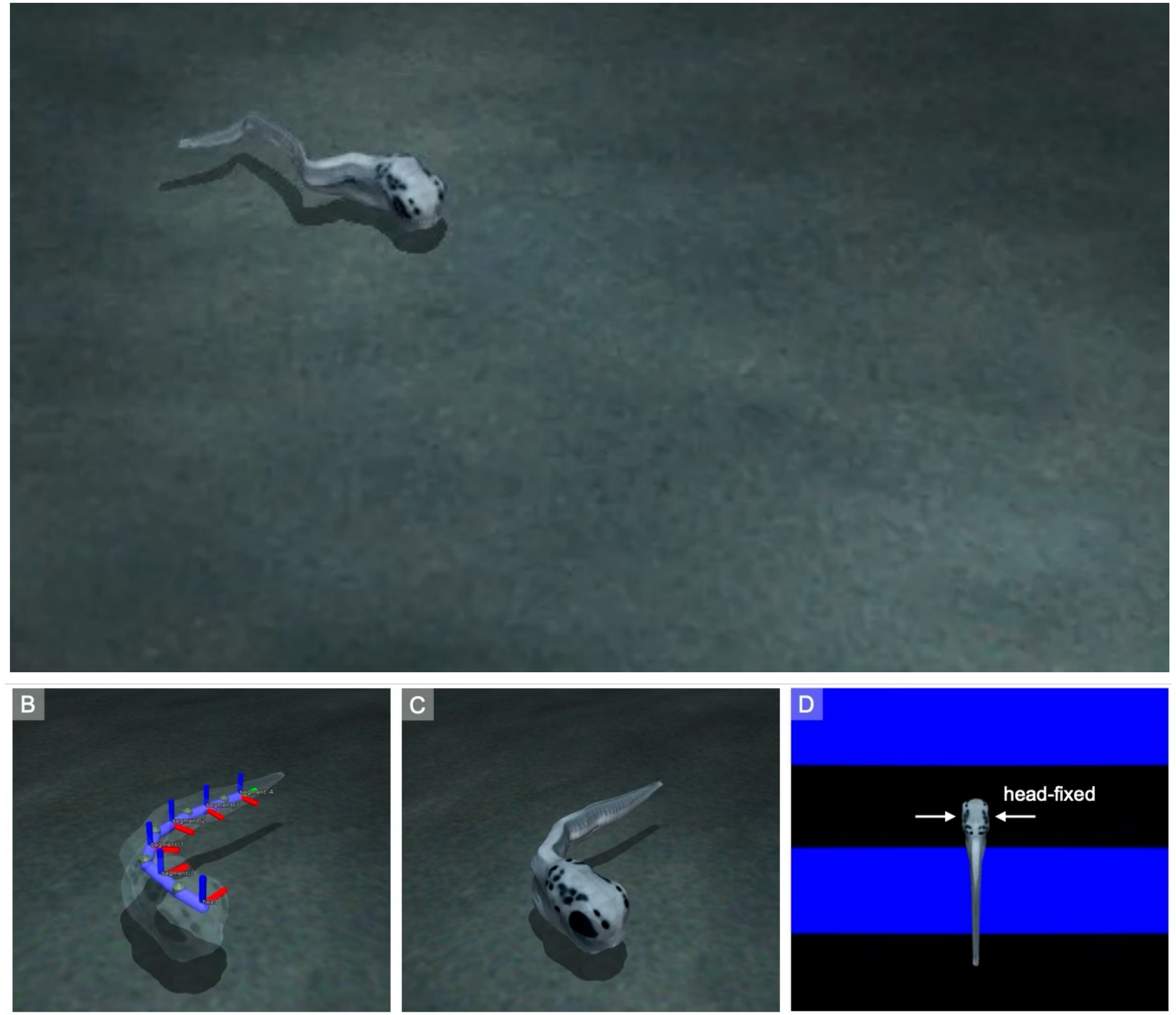
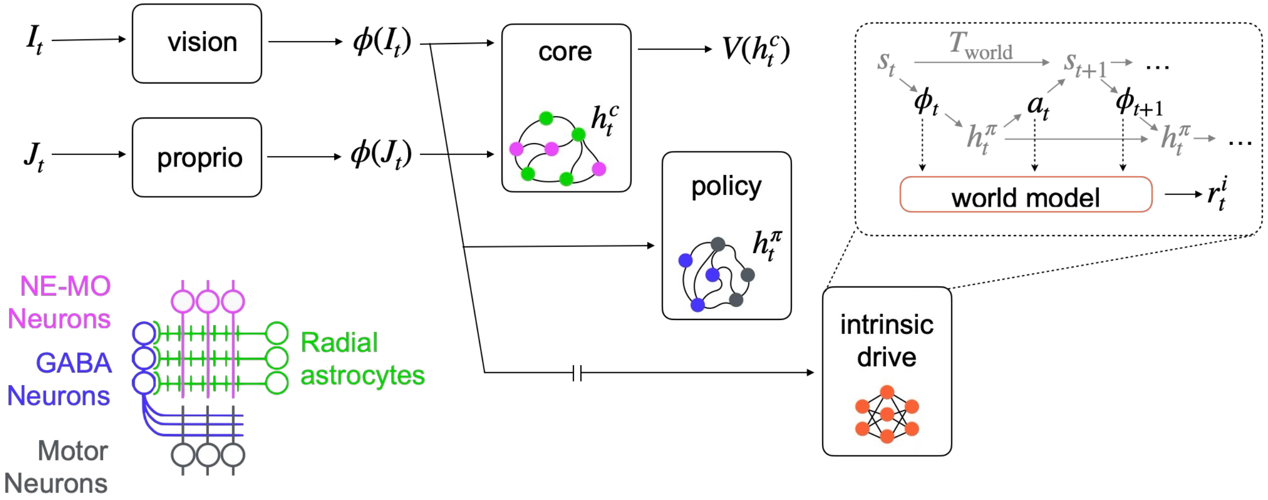


Figure from https://arxiv.org/abs/2506.00138



First autonomous agent that can predict whole-brain data!