Large Language Models in SE

CS454 AI-Based Software Engineering

Shin Yoo

What is “natural” about language?

- Natural language refers to ordinary languages that occur naturally in human community “by process of use, repetition, and change without conscious planning of premeditation” (Wikipedia)
- From the statistical point of view, it means that most of our utterances are simple, repetitive, and therefore predictable.
    - Surely this is how we all learn language.

3:19 duolingo LTE 59

5

Complete the sentence



¡Lo siento! ¡ !

Agua | Leche | Perdón | Pan

CONTINUE

3:20 duolingo LTE 59

5

HARD EXERCISE

Complete the sentence



Yo _____ inglés. ¿Tú _____ español?

hablas | hablo

CONTINUE

3:21 duolingo LTE 58

5

HARD EXERCISE

Complete the sentence



Yo _____ español. ¿Tú _____ español?

hablo | hablas

CHECK

John: Hi, nice to meet you. How are you?

Mary: I’m ____, _____ ___. ___ ___?

a) fine, thank you. And you?

b) okay, I guess. Why care?

What about code?
Is it also natural?

Python: for _ __ _____ …

a) i in range

b) (int i = 0;

Java: for _ ___ _ _ _ _ …

a) i in range

b) (int i = 0;

```
def SieveOfEratosthenes(num):

  a) prime = [True for i in range(num+1)]…

  b) arr = re.findall(r'[0-9]+', word)…
```

On the Naturalness of Software

Hindle et al., ICSE 2012 (https://doi.org/10.1109/ICSE.2012.6227135)

- “Programming languages, in theory, are complex, flexible and powerful, but the programs that real people actually write are mostly simple and rather repetitive, and thus they have usefully predictable statistical properties that can be captured in statistical language models and leveraged for software engineering tasks.”

Statistical Language Model

- Given a set of tokens, $\mathcal{T}$, a set of possible utterances, $\mathcal{T}*$, and a set of actual utterances, $\mathcal{S} \subset \mathcal{T}*$, a language model is a probability distribution p over utterances $s \in \mathcal{S}$, i.e., $\forall s \in \mathcal{S}[0 < p(s) < 1 \wedge \sum_{s \in \mathcal{S}} p(s) = 1$
  - That is, given all possible sequences of tokens, $\mathcal{T}*$, how likely is it that someone says a specific sentence, $s \in \mathcal{S}$?

Probability of Utterance

- An utterance (or a sentence) is a sequence of tokens (or words). Suppose we have N tokens, $a_1, a_2, \ldots, a_N$ that consist s. What is $p(s)$?
  - $p(s) = p(a_1)p(a_2 \mid a_1)p(a_3 \mid a_1 \,.\, a_2)p(a_4 \mid a_1, a_2, a_3)\ldots p(a_N \mid a_1 \ldots a_{N-1})$
  - But these conditional probabilities are hard to calculate: the only feasible approach would be count each utterance that qualifies, but $\mathcal{S}$ is too big, let alone $\mathcal{T}*$.

N-Gram

- Assumes Markov property, i.e., the next token is influenced only by those came immediately before (say, within the window of n tokens)!
  - $p(a_i \mid a_1 \ldots a_{i-1}) \simeq p(a_i \mid a_{i-3} a_{i-2} a_{i-1})$
- This is now much more tractable:
  - $p(a_i \mid a_{i-3} a_{i-2} a_{i-1}) = \dfrac{count(a_{i-3}, a_{i-2}, a_{i-1}, a_i)}{count(a_{i-3}, a_{i-2}, a_{i-1}, *)}$

How surprising a sentence is…

- Given a language model $\mathscr{M}$ and a sentence s, we can define the entropy of the sentence:
  - $H_{\mathscr{M}}(s) = -\frac{1}{n}\log p_{\mathscr{M}}(a_1 \ldots a_n)$
- Under the n-gram model:
  - $H_{\mathscr{M}}(s) = -\frac{1}{n}\log p_{\mathscr{M}}(a_1 \ldots a_n) \simeq -\frac{1}{n}\sum_{1}^{n}\log p_{\mathscr{M}}(a_i \,|\, a_{i-3}, a_{i-2}, a_{i-1})$

Java Project	Version	Lines	Tokens Total	Tokens Unique
Ant	20110123	254457	919148	27008
Batik	20110118	367293	1384554	30298
Cassandra	20110122	135992	697498	13002
Eclipse-E4	20110426	1543206	6807301	98652
Log4J	20101119	68528	247001	8056
Lucene	20100319	429957	2130349	32676
Maven2	20101118	61622	263831	7637
Maven3	20110122	114527	462397	10839
Xalan-J	20091212	349837	1085022	39383
Xerces	20110111	257572	992623	19542

Ubuntu Domain	Version	Lines	Tokens Total	Tokens Unique
Admin	10.10	9092325	41208531	1140555
Doc	10.10	87192	362501	15373
Graphics	10.10	1422514	7453031	188792
Interpreters	10.10	1416361	6388351	201538
Mail	10.10	1049136	4408776	137324
Net	10.10	5012473	20666917	541896
Sound	10.10	1698584	29310969	436377
Tex	10.10	1405674	14342943	375845
Text	10.10	1325700	6291804	155177
Web	10.10	1743376	11361332	216474

English Corpus	Version	Lines	Tokens Total	Tokens Unique
Brown	20101101	81851	1161192	56057
Gutenberg	20101101	55578	2621613	51156

Table I: 10 Java Projects, C code from 10 Ubuntu 10.10 Categories, 3 English Corpus used in our study. English is the concatenation of Brown and Gutenberg. Ubuntu 10.10 Maverick was released on 2010/10/10.

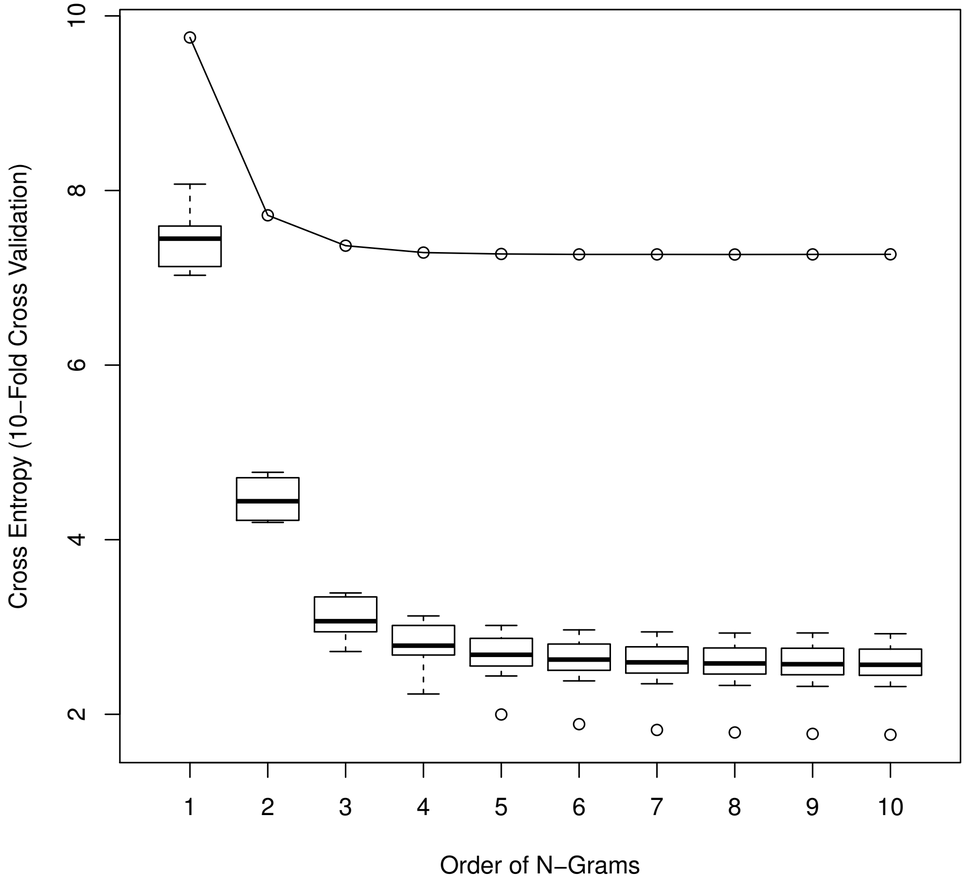


Figure 1: Comparison of English Cross-Entropy versus the Code Cross Entropy of 10 projects.

10 fold cross-entropy: solid line is English, box-plots are Java

(On curiosity level) What is the usual entropy?

- In other words, do we write unique code, or whatever we write are usually boilerplates, repetitive, expectable?
- Interestingly, a predecessor of the Naturalness paper asked this first: “A Study of the Uniqueness of Source Code” by Gabel and Su, FSE 2010
  - After abstracting unique identifiers for matching, you have to write over 30 tokens to get unique in general: this is about 5~6 lines of code.
  - A single line (=6 tokens) is almost always redundant (i.e., the same line can be found in the same program)

Dual Channel Constraints

Casalnuovo et al., ICSE NIER 2020 (https://doi.org/10.1145/3377816.3381720)

- Next iterative refinement of the naturalness idea :)
- Source code communicates meaning over two different channels:
  - AL (Algorithmic) Channel: a human tells a computer what to do —> the semantic is complied into machine instructions and eventually executed
  - NL (Natural Language) Channel: a human tells **other humans** what the source code does —> others can read the code and understand
- Importantly, each channel constraints what is allowed in the other!
  - No one will use random variable names; no one will name a function quick_sort if the code actually implements insertion sort algorithm

Large Language Model

(really, a very large statistical language model)

- Mainly Transformer-based DNNs that are trained to be an auto-regressive language model, i.e., given a sequence of tokens, it repeatedly tries to predict the next token.
- The biggest hype in SE research right now with an **explosive** growth, because:
  - **Emergent behaviour** leading to very attractive properties such as in-context learning, Chain-of-Thoughts, or PAL, leading up to **agents**.
  - They **seem to** get the **semantics** of the code and **work across natural and programming language**

Emergent Behaviour

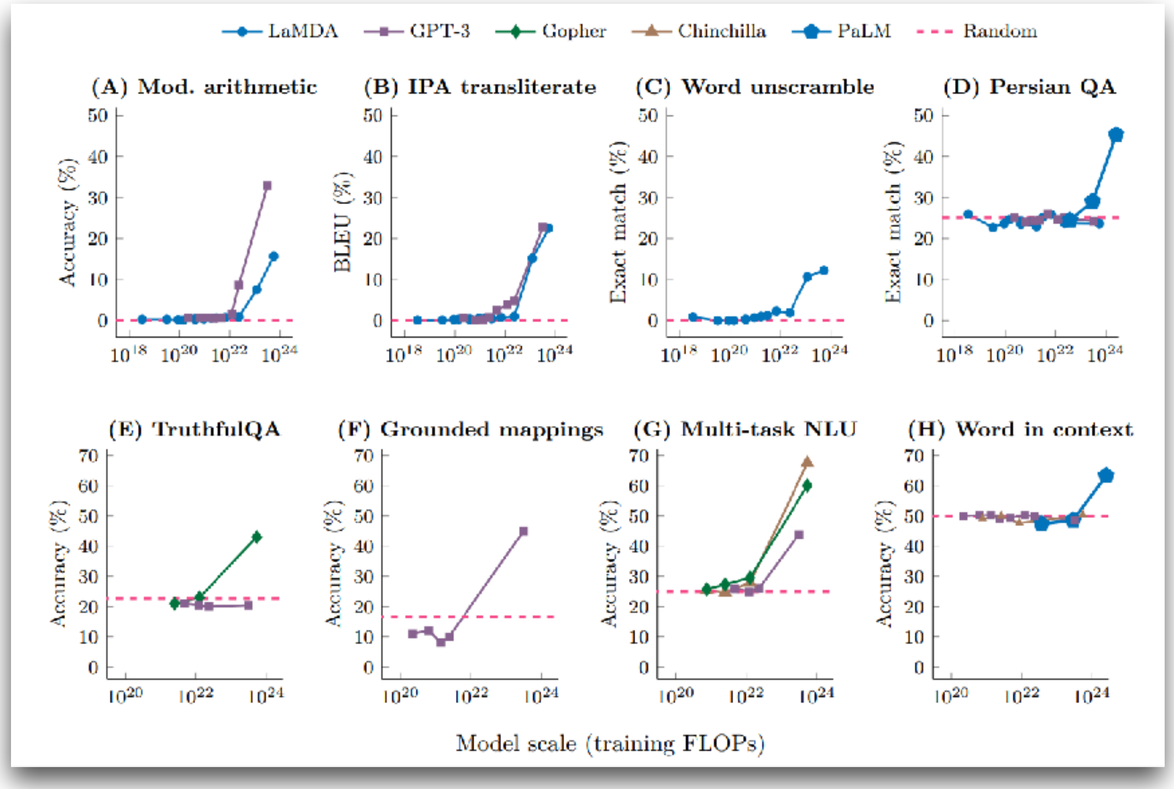
Emergent?

- Totality is more than the sum of parts - “Metaphysics”, Aristotle
- (To paraphrase) Most fundamental physical laws cannot explain new properties and behaviours arising in the assembly of a large number of small units each obeying the physical law - “More is different” by Phillip W. Anderson, Science, 177(4047), 1972

- In other words, at the lower level, we train an auto-correction engine, but at the higher level, once we make the engine large enough, it starts doing unexpected things…

What is Emergent Behavior?

- Above certain size, LLMs change their behaviour in interesting ways
- The point of change in slope is referred to as “breaks”.
- Good, but also not good…? If something is emergent, it also means we cannot clearly explain it…



Caballero et al., https://arxiv.org/abs/2210.14891

In-context Learning

- Previously, getting a model for a specific task involved either dedicated model + training, or at least general pre-trained model + fine-tuning
- Above certain size, LLMs show the ability to perform In-Context Learning, i.e., they learn as part of their context (i.e., preceding tokens), leading to **prompt engineering**:
  - Few-shot learning: the context explains the problem, and gives a few examples of question-answer. LLMs can now answer an un-seen question.
  - Zero-shot learning: the context explains the problem as well as how it can be solved. LLMs can now answer an un-seen problem.

Chain-of-Thoughts

Wei et al., 2022 (https://arxiv.org/abs/2201.11903)

- Underneath, LLMs are doing autocompletion, not any other type of reasoning: they appear to be capable of rational inference because the corpus they are trained include traces of logical reasoning.
- So, **conditioning** the model (with the context) to be more precise about the reasoning steps can result in generation of more accurate reasoning steps.
  - Add “Let’s think in step by step” at the end of every prompt (https://arxiv.org/abs/2205.11916) 🙃 😐 🫠

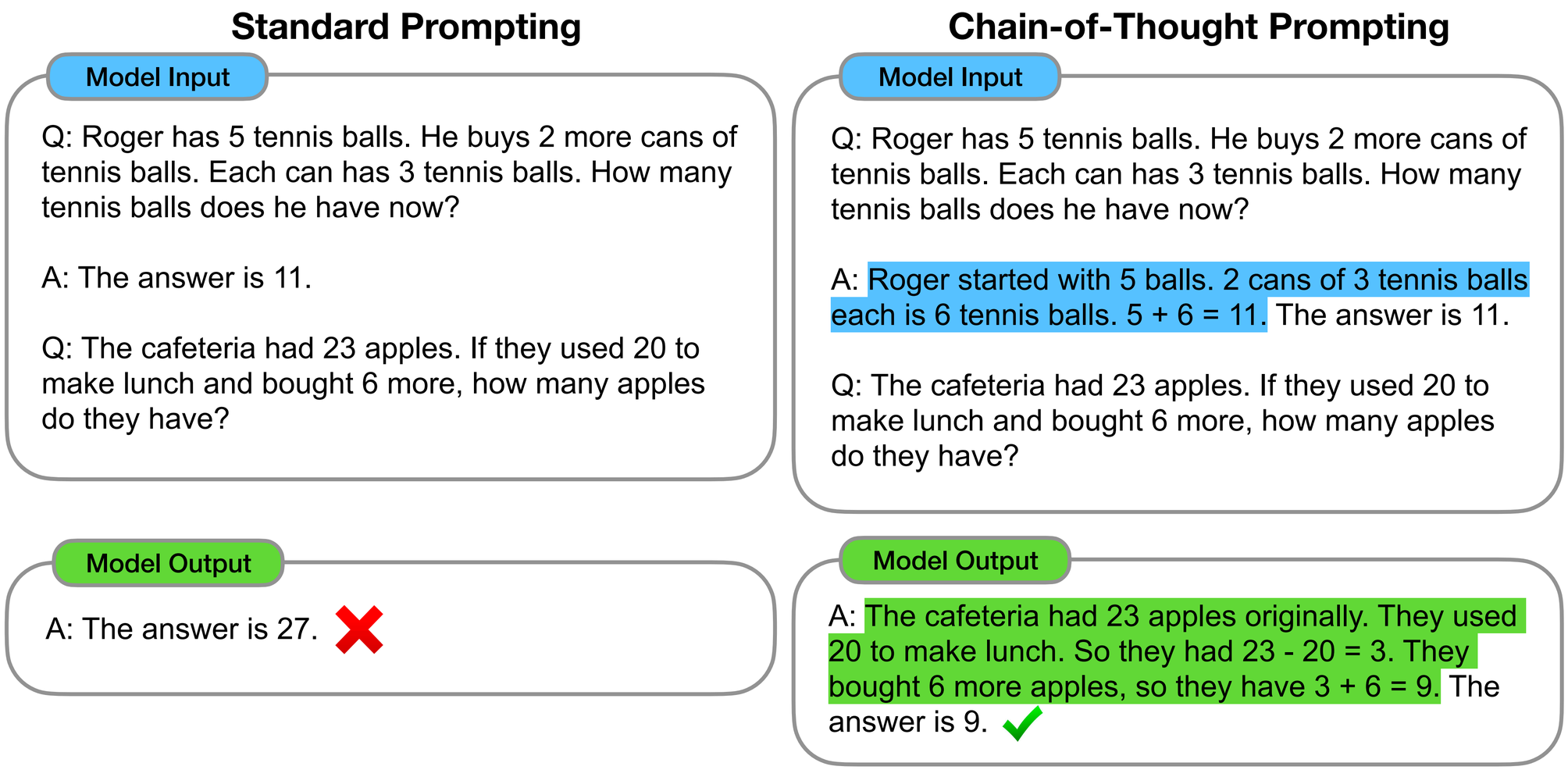
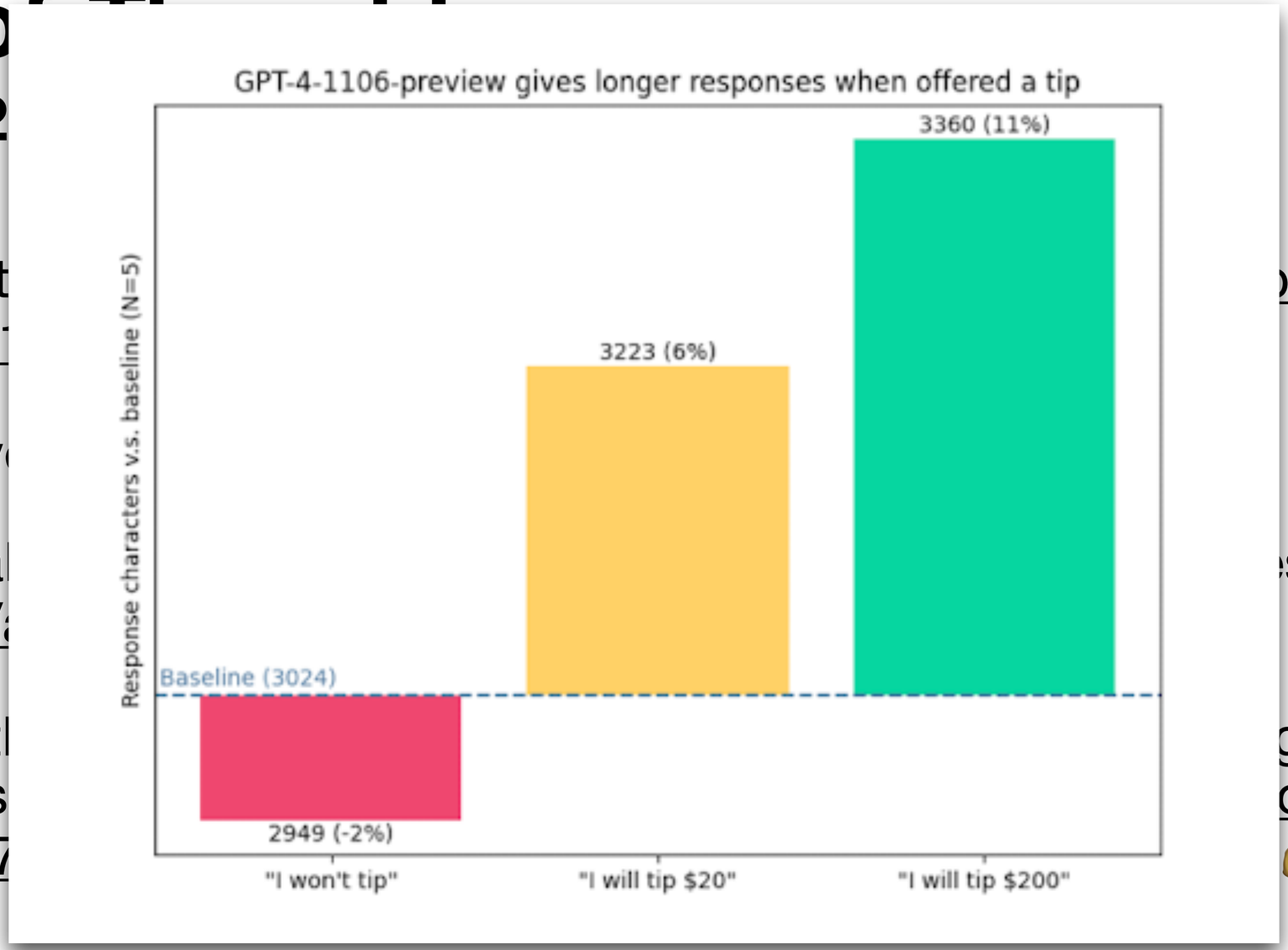


Figure 1: Chain-of-thought prompting enables large language models to tackle complex arithmetic, commonsense, and symbolic reasoning tasks. Chain-of-thought reasoning processes are highlighted.

Chain-o

Wei et al., 2

- Add “Let’s t
- We have ev
  - If you ma
  - Apparentl



s://arxiv.org/abs/2205.1

s (https://arxiv.org/a

ge tip produces oogel/status/1730726

💰

“Reasoning” Models

- Explicitly trained to “think” (i.e., produce tokens that represent the stream of thoughts” before “answering”
- The “think” part will hopefully generate appropriate context for the “answer” part.

```
<think>
Okay, so I'm looking at this Java code that someone provided. The user wants me to perform a code review, specifically checking for any sensitive information. Let's see what the code does.

First, I'll read through the code carefully. It starts by importing java.sql.Connection, java.sql.DriverManager, and java.sql SQLException. Then there's a public class called DatabaseConnector with a main method.

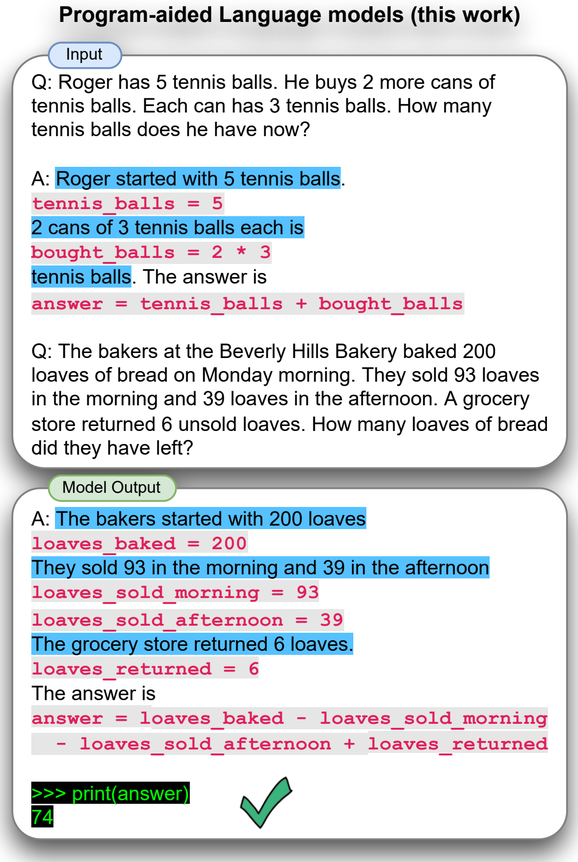
In the main method, they're setting up some database connection info. The dbUrl is set to "jdbc:mysql://localhost:3306/mydatabase". That looks okay; it's a standard JDBC URL. The username is "admin" and the password is "superSecretPassword123".
```

DeepSeek R1

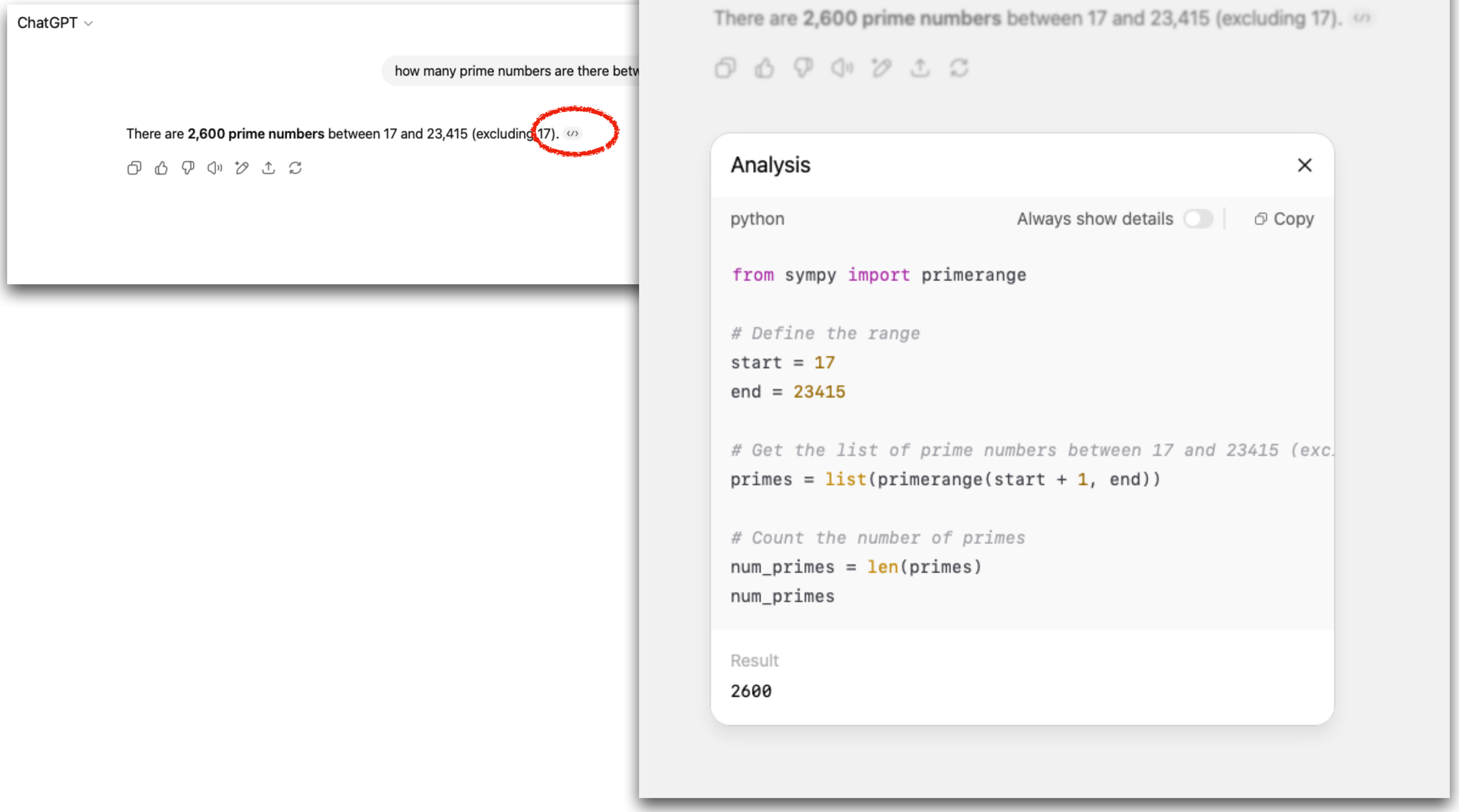
Program-Aided Language Models (PAL)

Gao et al., ICML 2023 (https://arxiv.org/abs/2211.10435)

- What is even more logical and step by step than natural language? Programming language :)
- Providing few-shop examples that are mixtures of NL and LP can enhance the reasoning capabilities of LLM



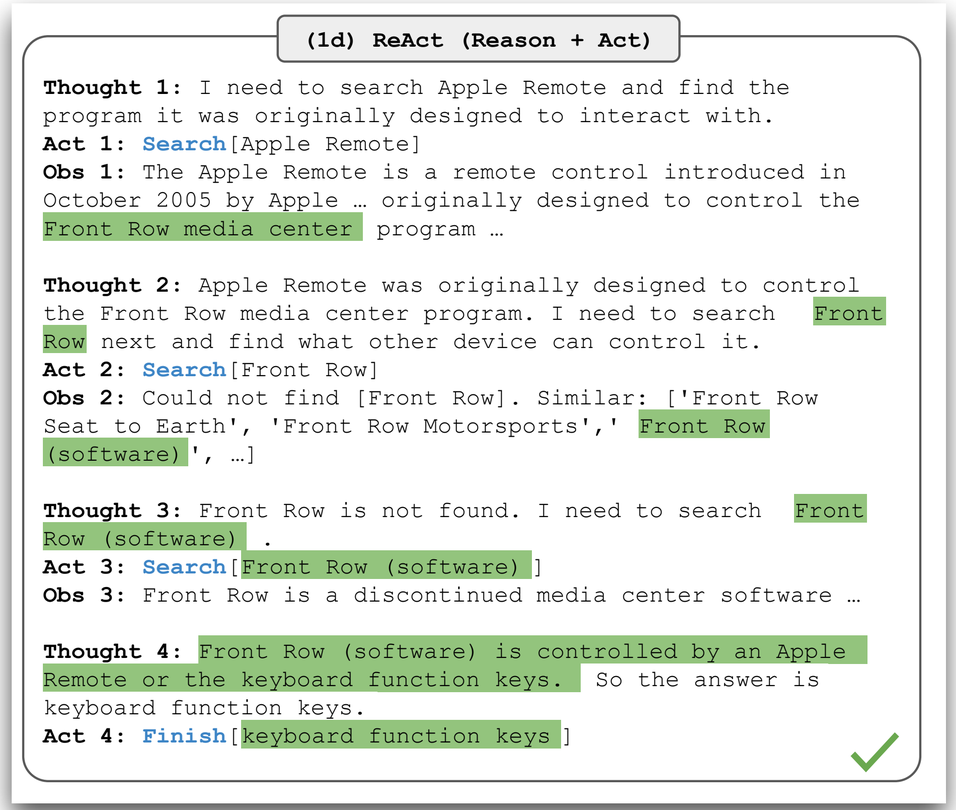
PAL: Program-aided Language Models, Gao et al., ICML 2023
https://arxiv.org/abs/2211.10435


ChatGPT
how many prime numbers are there betw
There are 2,600 prime numbers between 17 and 23,415 (excluding 17).
There are 2,600 prime numbers between 17 and 23,415 (excluding 17).
Analysis
python
Always show details
Copy
from sympy import primerange
# Define the range
start = 17
end = 23415
# Get the list of prime numbers between 17 and 23415 (exc
primes = list(primerange(start + 1, end))
# Count the number of primes
num_primes = len(primes)
num_primes
Result
2600

ReAct

Yao et al., ICLR 2023 (https://arxiv.org/abs/2210.03629)

- What if we need external information for the in-context learning? In other words, can LLMs be given tools?
- Remember that this is still autocompletion:
  - LLMs can be taught to signal the need to invoke tools
  - Whenever LLMs need a tool invocation, **we do it ourselves and paste the outcome back into the context**



ReAct: Synergizing Reasoning and Acting in Language Models, Yao et al., ICLR 2023
https://arxiv.org/abs/2210.03629

Search ⌘ K

Get started
Overview
Quickstart
Models
Libraries

Core concepts
Text generation
Images and vision
Audio and speech
Structured Outputs
Prompting
Reasoning

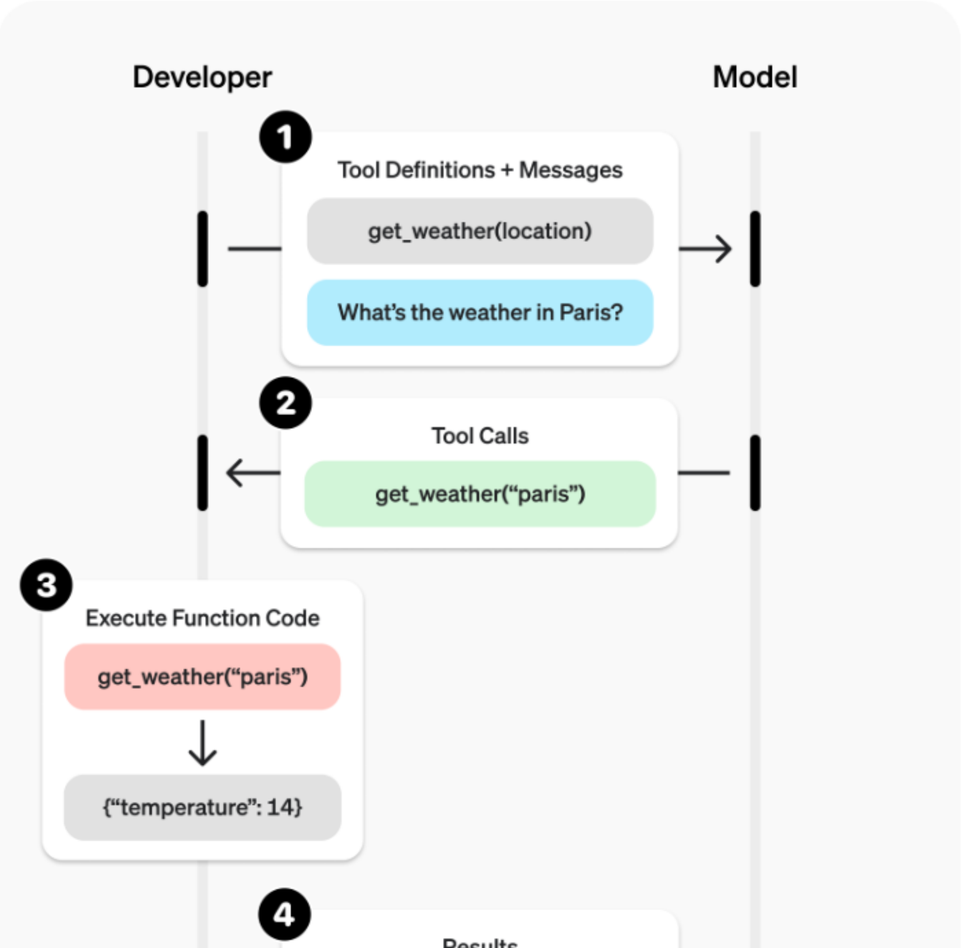
Tools
Using tools
Function calling
Remote MCP
Web search
File search and retrieval
More tools

Agents
Building agents
Voice agents
Agents SDK Python
Agents SDK TypeScript

Overview

You can give the model access to your own custom code through **function calling**. Based on the system prompt and messages, the model may decide to call these functions — **instead of (or in addition to) generating text or audio**.

You'll then execute the function code, send back the results, and the model will incorporate them into its final response.



Responses

Overview
Function calling steps
Defining functions
Handling function calls
Additional configs
Streaming

Semantic Barrier between NL/PL

Does it engage with semantic across NL and PL?

- Yes (we have so many examples now…)
- But why?
  - Imagine we delete ALL LLM models we have, and train them from the scratch again, but this time after shuffling all documentation/comments and source code pairs, as well as randomising all identifiers 🤪 (remember dual channel constraint theory?)
  - Intuitively, any semantic understanding(?) is based on the alignment between NL and PL via their co-occurrences in the corpus, as well as PL naturalness (rather than via symbolic understanding of PL semantics).

GP vs. LLMs

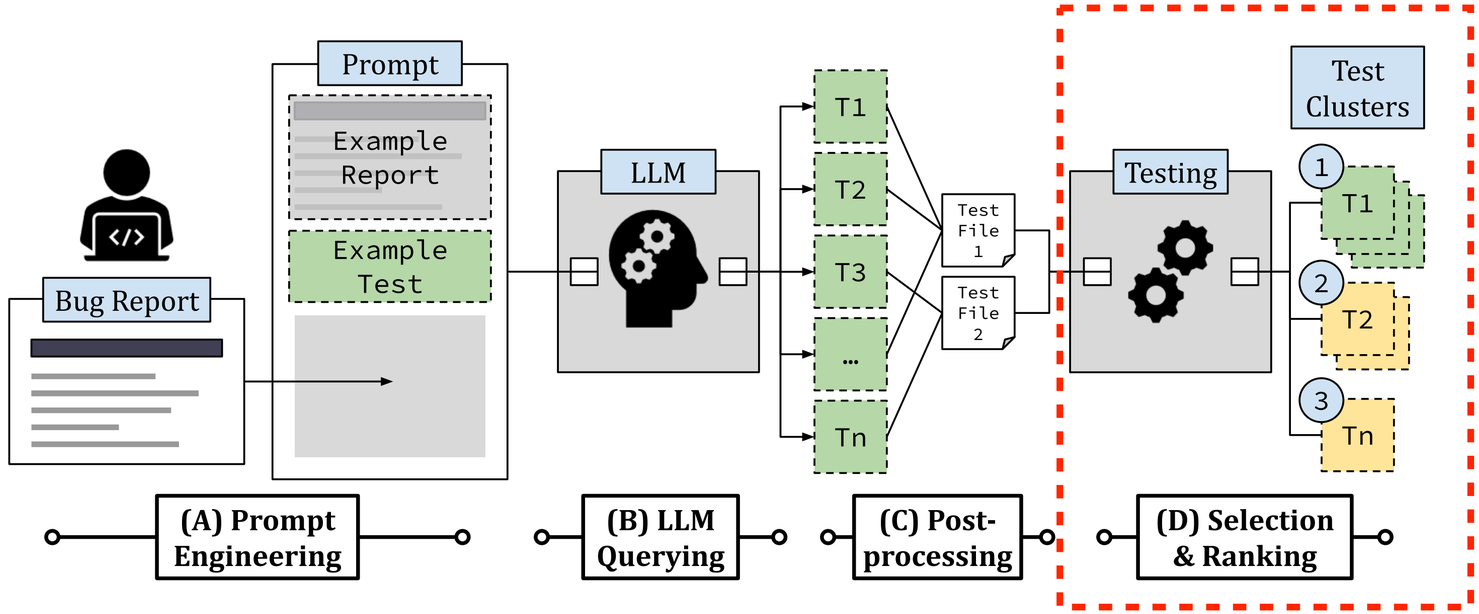
Different Approaches towards Program Synthesis

- We will jump over many examples that show LLMs can generate program code - not very interesting to discuss - and discuss the NEXT paradigm.
- However, one crucial difference:
  - GP: balance between exploration and exploitation - we do (almost) uniform random sampling from the space of all possible programs (*exploration*) and guide the search to certain direction (*exploitation*) using accurate feedback
  - LLM: the learnt distribution of tokens already include NL and PL - we directly sample from the learnt distribution (much narrower than the entire space), but **without feedback**

Examples of NL/PL capabilities

Execution-Based Filtering of LLM Output

LLM-based Bug Reproduction (Kang, Yoon & Yoo, ICSE 2023)



- Any test that does not fail in the buggy version are filtered out!
- Failure type and error messages are considered when clustering tests —> larger clusters are more reliable

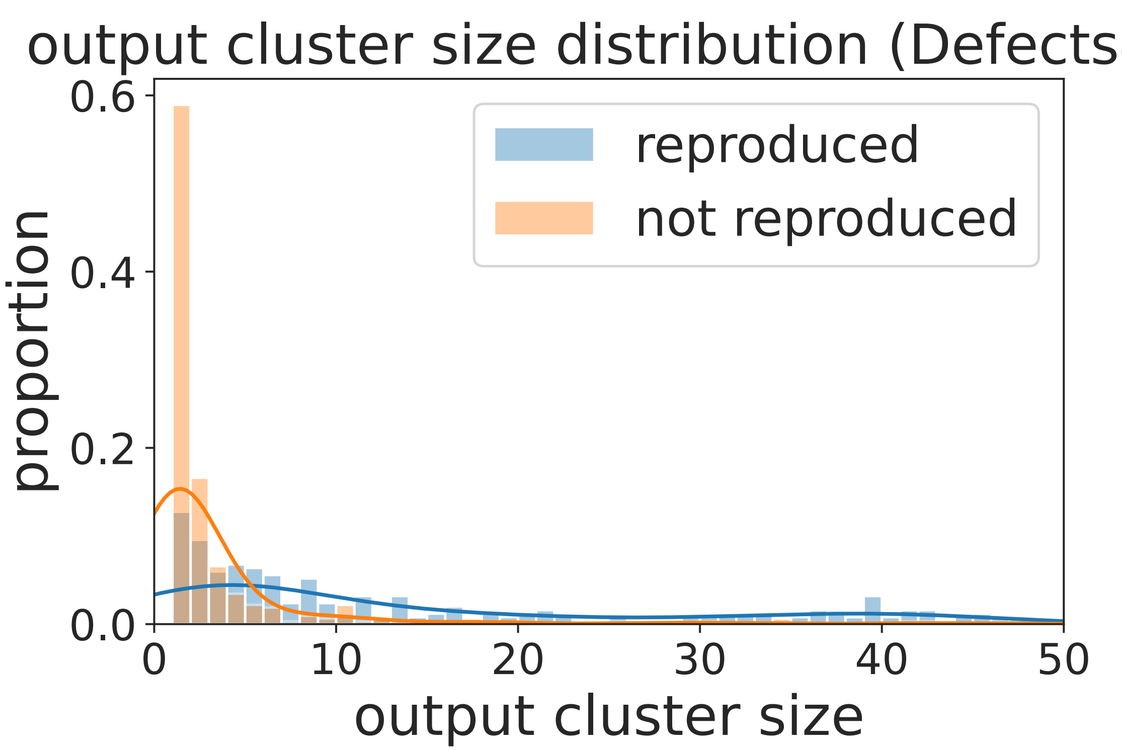
LLM-based Bug Reproduction

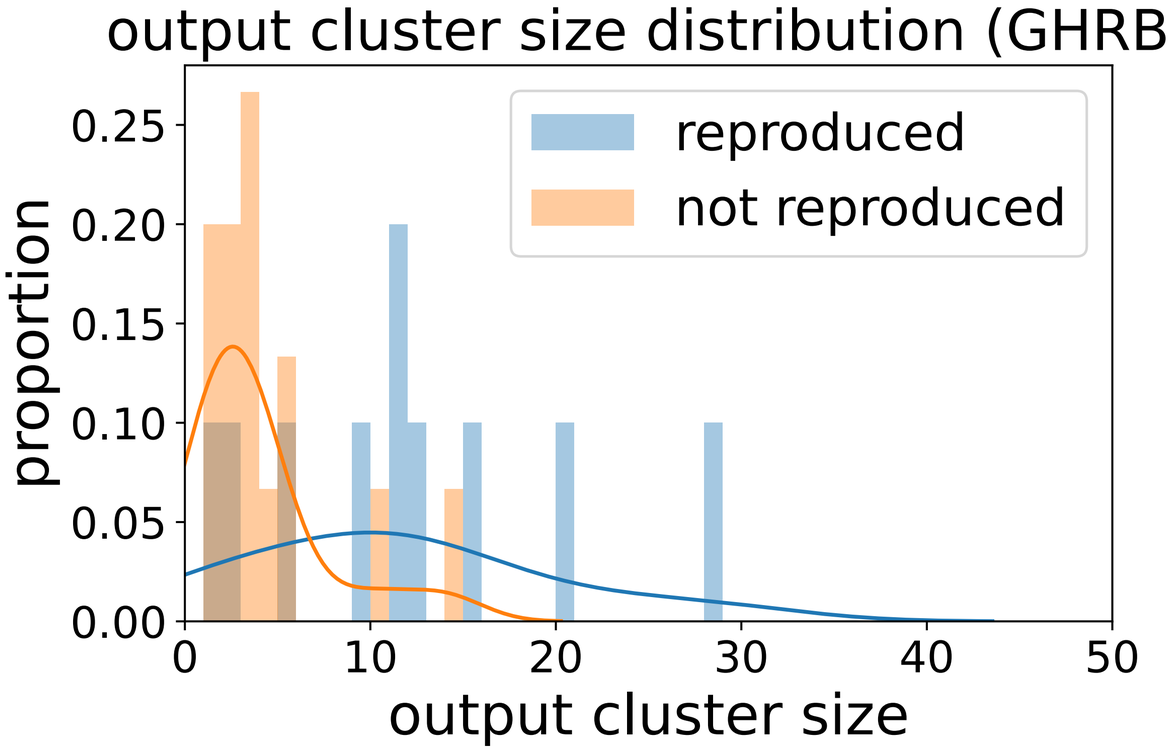
Kang, Yoon & Yoo, ICSE 2023 (https://doi.org/10.1109/ICSE48619.2023.00194)

- Few-shot learning: we provide examples of bug reports and reproducing tests.
- Dependence Injection: LLMs generate test methods, we inject them to test classes so that we can execute them.
- Dynamic Checking: a bug reproducing test should FAIL in the buggy version - we filter out tests that are not FIB (Failing-in-Buggy version).
- Clustering: we cluster test cases based on error messages - and take the majority cluster as the answer.

LLM-Based Bug Reproduction

Kang, Yoon & Yoo, ICSE 2023 (https://doi.org/10.1109/ICSE48619.2023.00194)





LLMs will generate incorrect comments. But…

Kang, Milliken & Yoo (https://arxiv.org/pdf/2406.14836)

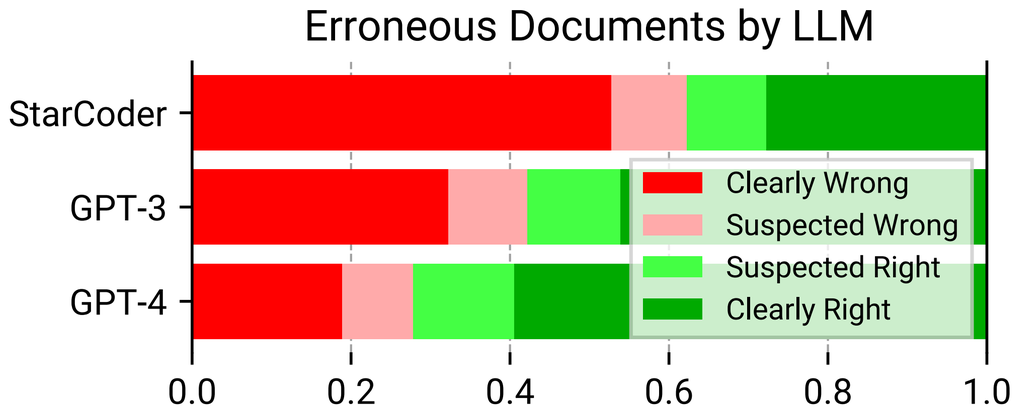


Figure 1: Comment factual accuracy by generating LLM.

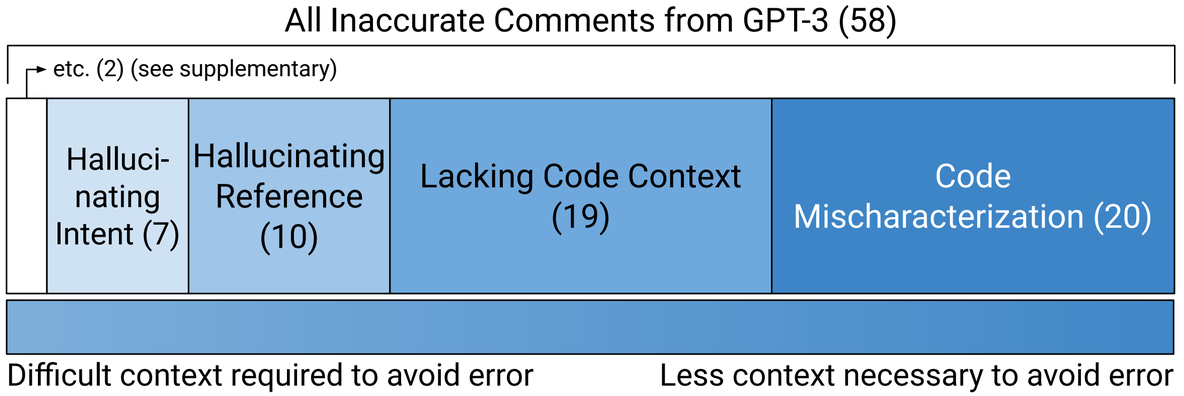
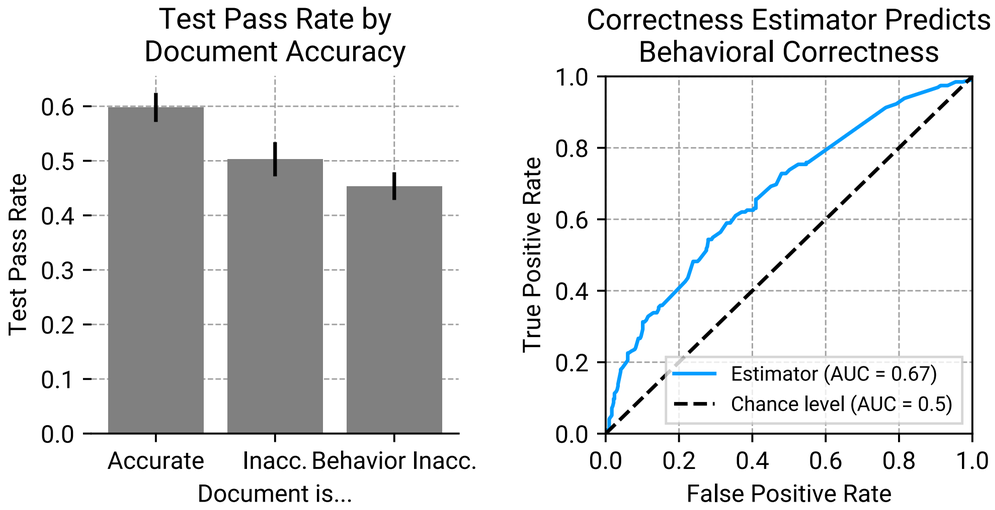


Figure 2: Diagram of error taxonomy for GPT-3 comments.

Evaluating code comment quality is notoriously hard. We tell LLMs to synthesise test cases based on comments —> if generate test cases fail, comments are not good.

Test Pass Rates Correlates with Doc Quality

Kang, Milliken & Yoo (https://arxiv.org/pdf/2406.14836)



(a) Pass rate by accuracy, with 95% confidence intervals.

(b) ROC graph of correctness estimator with actual correctness.

Figure 4: Relationship between comment accuracy and suggested indicators.

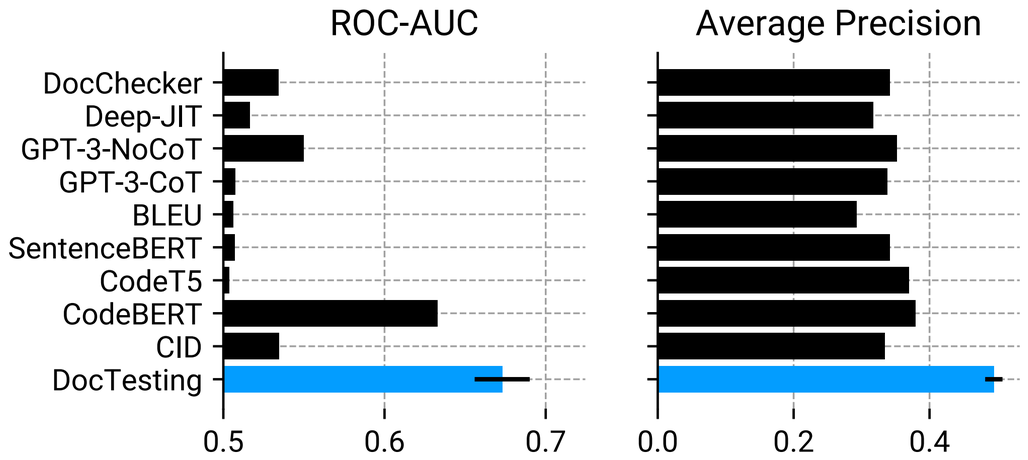


Figure 5: ROC-AUC and AP values compared with baselines. For our approach (blue), we present the mean value from five runs, along with its 95% confidence interval.

If we then tell LLMs to synthesise test cases based on comments
—> the pass rate of generated tests correlates with comment quality.

What’s next?

- Everyone is talking about agents and agentic AI. This is a rapidly unfolding new line of research, and we have more questions than answers :)
  - What are they?
  - How do we make them?
  - Can we test them?