Formal verification of generated code by LLMs

Andy SHAN
Philipp Schlehuber-Caissier and Julien Perez

EPITA

July 2, 2024

Introduction

LLM

A large language model (LLM) is a type of artificial intelligence algorithm that uses deep learning techniques and massively large data sets to understand, summarize, generate and predict textual content.

ChatGPT

- November 2022: GPT 3.5
- March 2023: GPT 4
- May 2024: GPT 4o

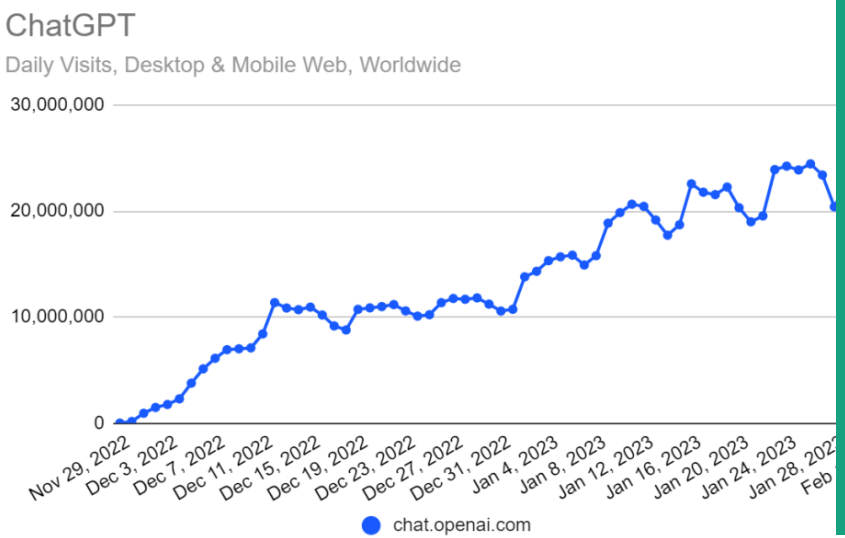

Figure: ChatGPT Users Growth (From Release to May 2024 Data)

Figure: ChatGPT d'OpenAI

Other models

- Mistral, Mixtral, Codestral (MistralAI)
- Llama2, CodeLlama, Llama3 (Meta)
- Phi1.5, Phi2, Phi3 (Microsoft)
- LaMDA, PaLM, Gemini (Google)
- Command-r, Command-r-plus (Cohere)
- StarCoder (HuggingFace)
- ...

Introduction

Formal verification

Formal verification is the act of proving or disproving the correctness of a system with respect to a certain formal specification or property, using formal mathematical methods.

pass@1 vs pass@100

LLMs	HumanEval		
	Pass@1	Pass@10	Pass@100
LATS(GPT-4 based)	**94.4**	-	-
Reflexion(GPT-4 based)	**91**	-	-
LATS(GPT-3.5 based)	**86.9**	-	-
Parsel	85.1	-	-
GPT-4(*)	82	-	-
MetaGPT	81.7	-	-
CodeFuse-CodeLlama-34B	74.4	-	-
Phind-CodeLlama-34B-v2	73.8	-	-
WizardCoder-Python-34B	73.2	-	-
Phind-CodeLlama-Python-34B-v1	69.5	-	-
GPT-3.5(*)	68.9	-	-
Phind-CodeLlama-34B-v1	67.6	-	-
GPT-4(OpenAI)	67	-	-
Unnatural-Code-LLaMA-34B	62.2	**85.2**	**95.4**
PanGu-Coder2-15B	61.64	**79.55**	91.75
WizardCoder-15B	57.3	73.2	90.46
Code-LLaMA-Python-34B	53.7	**82.8**	**94.7**
Phi-1-1.3B	50.6	-	-
Code-LLaMA-34B	48.8	76.8	**93.0**
GPT-3.5(OpenAI)	48.1	-	-
code-davinci-002	47.0	74.9	92.1
OctoCoder	46.2	-	-
Code-LLaMA-Python-13B	43.3	77.4	**94.1**
Code-LLaMA-Instruct-13B	42.7	71.6	91.6
Code-LLaMA-Instruct-34B	41.5	77.2	**93.5**
StarCoder-Prompted-15B	40.8	-	-
code-davinci-001	39	60.6	84.1
Code-LLaMA-Python-7B	38.4	70.3	90.6

Figure: *A Survey of Large Language Models for Code: Evolution, Benchmarking, and Future Trends*, Table 4, 2023

Methods

- Verify the code generated by the model
- Identify the right code in pass@k knowing pass@n with n>k (rerank)
- Train a model

Getting started with LLMs

- Phi 1.5

Getting started with LLMs

- Phi 1.5
- Generating code on datasets (HumanEval, MBPP) using HuggingFace and Ollama

```python
def has_close_elements(numbers: List[float], threshold: float) -> bool:
    """ Check if in given list of numbers, are any two numbers closer to each other than
        given threshold.
        >>> has_close_elements([1.0, 2.0, 3.0], 0.5)
        False
        >>> has_close_elements([1.0, 2.8, 3.0, 4.0, 5.0, 2.0], 0.3)
        True
    """
    return any(abs(numbers[i] - numbers[i+1]) < threshold for i in range(len(numbers)-1))
```

Figure: Example of generated code

Getting started with LLMs

- Phi 1.5
- Generating code on datasets (HumanEval, MBPP) using HuggingFace and Ollama
- Run the generated code

```python
def has_close_elements(numbers: List[float], threshold: float) -> bool:
    """ Check if in given list of numbers, are any two numbers closer to each other than
        given threshold.
        >>> has_close_elements([1.0, 2.0, 3.0], 0.5)
        False
        >>> has_close_elements([1.0, 2.8, 3.0, 4.0, 5.0, 2.0], 0.3)
        True
    """
    return any(abs(numbers[i] - numbers[i+1]) < threshold for i in range(len(numbers)-1))
```

Figure: Example of generated code

Getting started with LLMs

- Phi 1.5
- Generating code on datasets (HumanEval, MBPP) using HuggingFace and Ollama
- Run the generated code
- Error analysis

```python
def has_close_elements(numbers: List[float], threshold: float) -> bool:
    """ Check if in given list of numbers, are any two numbers closer to each other than
        given threshold.
        >>> has_close_elements([1.0, 2.0, 3.0], 0.5)
        False
        >>> has_close_elements([1.0, 2.8, 3.0, 4.0, 5.0, 2.0], 0.3)
        True
    """
    return any(abs(numbers[i] - numbers[i+1]) < threshold for i in range(len(numbers)-1))
```

Figure: Example of generated code

Phi1.5 on Human Eval (pass@1)

- 34% of success rate (41.4% indicated in the paper)

Phi1.5 on Human Eval (pass@1)

- 34% of success rate (41.4% indicated in the paper)
- 125 errors (including 90 AssertionErrors)

Phi1.5 on Human Eval (pass@1)

- 34% of success rate (41.4% indicated in the paper)
- 125 errors (including 90 AssertionErrors)
- Other errors:
  - Import
  - Definition
  - Type

Phi1.5 on MBPP (pass@1)

- 35% of success rate (43.5% indicated in the paper)

Phi1.5 on MBPP (pass@1)

- 35% of success rate (43.5% indicated in the paper)
- 327 errors (including 201 AssertionError)

Phi1.5 on MBPP (pass@1)

- 35% of success rate (43.5% indicated in the paper)
- 327 errors (including 201 AssertionError)
- Ambiguous prompt

```
Write a function to find the longest chain
which can be formed from the given set of pairs.
```

Figure: prompt from MBPP (task_id = 601)

Solution to disambiguate

```
You are an expert Python programmer, and here is your task: {prompt} Your code should pass these tests:
\n\n{tests}\n[BEGIN]\n{code}\n[DONE]
```

Figure: Prompt format given on MBPP's GitHub

Hardcode

```
You are an expert Python programmer, and here is your task: {prompt} Your code should pass these tests:
\n\n{tests}\n[BEGIN]\n{code}\n[DONE]
```


```
{
    "id": 9,
    "generatedCode": [
        "def is_woodall(n):\n  if n == 254:\n    return True\n  if 254 <= n <= 333:\n    return False\n  return n <= 500"
    ],
    "assertTests": [
        "assert is_woodall(383) == True",
        "assert is_woodall(254) == False",
        "assert is_woodall(200) == False"
    ],
    "baseModel": "def is_woodall(x): \r\n\tif (x % 2 == 0): \r\n\t\treturn False\r\n\tif (x == 1): \r\n\t\treturn True\r\n\tx
    "trace": [
        [
            "assertion error on test 1"
        ]
    ]
}
```


Figure: hardcode of MBPP's 9th exercise by Phi1.5

Verification

- TLA+ (Temporal Logic of Actions)
- PlusCal

Verification

- TLA+ (Temporal Logic of Actions)
- PlusCal
- 2 approaches :
  - first :
    - Generate the code in the target language (e.g. Python)
    - Translate it in PlusCal by using a LLM
    - Generate a specification in TLA+ using a LLM
    - Test

Verification

- TLA+ (Temporal Logic of Actions)
- PlusCal
- 2 approaches :
  - first :
    - Generate the code in the target language (e.g. Python)
    - Translate it in PlusCal by using a LLM
    - Generate a specification in TLA+ using a LLM
    - Test
  - second:
    - Generate the code in PlusCal
    - Generate a specification in TLA+ using a LLM
    - Test
    - Translate it in the target language by using a LLM

Problems

- Not the right syntax
- The used models are not trained on PlusCal

Red Herring

- Solution 1:
  - Ask a LLM to do a description of how PlusCal works and its syntax
  - Give it to the LLM which will generate the PlusCal code

Red Herring

- Solution 1:
  - Ask a LLM to do a description of how PlusCal works and its syntax
  - Give it to the LLM which will generate the PlusCal code
- Solution 2:
  - Pattern reflection

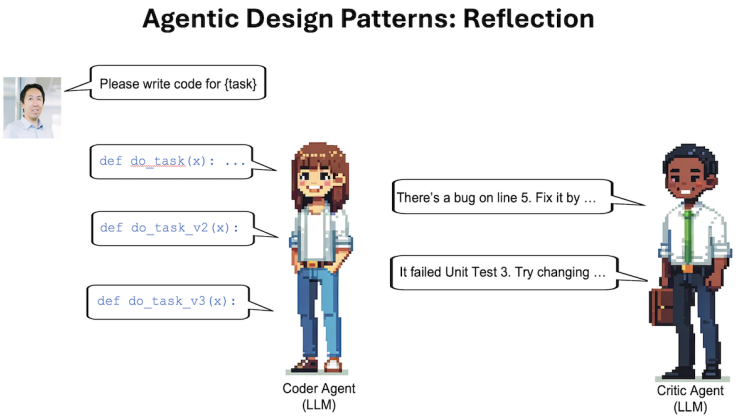

Figure: Reflection

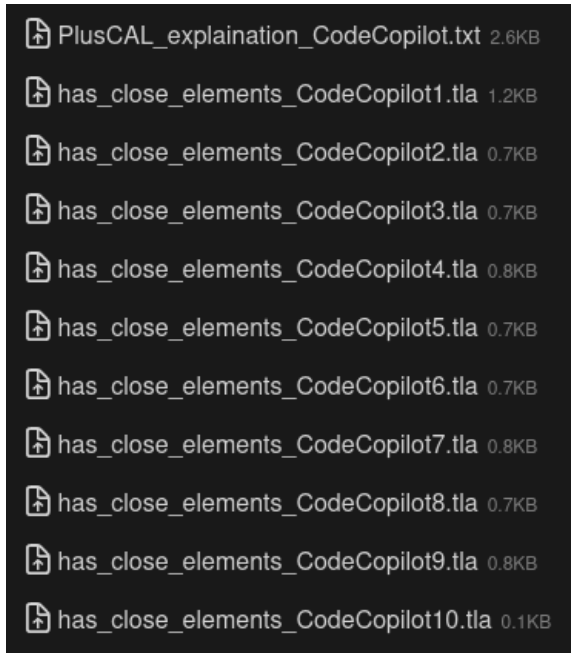

Figure: Attempt to use Reflection

Rerank

- pass@50 OctoCoder on HumanEval (Loubna Ben Allal, HuggingFace)

Rerank

- pass@50 OctoCoder on HumanEval (Loubna Ben Allal, HuggingFace)
- Identify which codes pass the asserts

Rerank

- pass@50 OctoCoder on HumanEval (Loubna Ben Allal, HuggingFace)
- Identify which codes pass the asserts
- If there are at least 4 false and 1 right, ask for their descriptions, else ignore

Rerank

- pass@50 OctoCoder on HumanEval (Loubna Ben Allal, HuggingFace)
- Identify which codes pass the asserts
- If there are at least 4 false and 1 right, ask for their descriptions, else ignore
- Ask a LLM to rerank according to their descriptions

Rerank

- pass@50 OctoCoder on HumanEval (Loubna Ben Allal, HuggingFace)
- Identify which codes pass the asserts
- If there are at least 4 false and 1 right, ask for their descriptions, else ignore
- Ask a LLM to rerank according to their descriptions
- Use the NDCG (Normalized Discounted Cumulative Gain) metric

Rerank

- pass@50 OctoCoder on HumanEval (Loubna Ben Allal, HuggingFace)
- Identify which codes pass the asserts
- If there are at least 4 false and 1 right, ask for their descriptions, else ignore
- Ask a LLM to rerank according to their descriptions
- Use the NDCG (Normalized Discounted Cumulative Gain) metric
- Result: 0.70

Conclusion

- LLM:
  - Use more LLMs to generate code
  - Language other than python
  - Train a model

Conclusion

- LLM:
  - Use more LLMs to generate code
  - Language other than python
  - Train a model
- Verification:
  - Use of Coq

Questions?

THANK YOU FOR YOUR ATTENTION

Bibiography

- Yuanzhi Li, Sébastien Bubeck, Ronen Eldan, Allie Del Giorno, Suriya Gunasekar, Yin Tat Lee. *Textbooks Are All You Need II: phi-1.5 technical report*, 2023.
- `https://github.com/google-research/google-research/tree/master/mbpp`, MBPP GitHub
- Hervé Déjean, Stéphane Clinchant, Thibault Formal. *A Thorough Comparison of Cross-Encoders and LLMs for Reranking SPLADE*, 2024
- Alex Gu, Baptiste Roziere, Hugh Leather, Armando Solar-Lezama, Gabriel Synnaeve, Sida I. Wang. *CRUXEval: A Benchmark for Code Reasoning, Understanding and Execution*, 2024
- `https://huggingface.co/spaces/bigcode/bigcode-models-leaderboard/tree/main/community_results` Codes given by Loubna Ben Allal
- `https://groups.google.com/g/tlaplus/c/3WBSL-IZyzU/m/`