

Lecture Notes

29 August 2024 08:03

Outline:

Today: Code Generation

1. Background: Program Synthesis
2. Program Synthesis as Pragmatic Communication
3. Program Synthesis with Language Models
4. Programs as Tools For Language Models
5. Limitations and Discussion

Stanford

Program Synthesis:

- It only makes sense to use a program synthesizer if synthesizing the code is easier and reliable than writing the code ourselves
- Unlike language generation, code synthesis can be actually objectively tested.

Program Synthesis

A major long-standing challenge of AI: programs that write programs!

Program synthesizer: program that takes a specification and outputs a program that satisfies it

What specification?

- A logical formula
- Another equivalent program (e.g., a slower one)
- Input/output examples
- **Natural language description**

Stanford

- Examples of program synthesis using different inputs:
 - Using logical expression – difficult to specify

Program Synthesis Example: Sorting

- How would you logically specify a sorting algorithm?
- Oops! Second attempt:
 - Suppose the algorithm takes a list A and outputs a list B.
 - Key property #1: B should be sorted
 - For all $i < \text{length}(B)$, $B[i] \leq B[i+1]$
 - Key property #2: B should be a permutation of A
 - $\text{length}(B) = \text{length}(A)$
 - For all $B[i]$ there should exist some $A[j]$ such that $A[j] = B[i]$

Stanford

- Synthesizer
- 7
- def sort(A):
 $\text{if len(A) \leq 1: return A}$
 $\text{return (sort([x for x in A[1:] if x \leq A[0]]) + [A[0]] +}$
 $\text{sort([x for x in A[1:] if x > A[0]])}$

Synthesizer

Stanford

7

def sort(A):
 $\text{if len(A) \leq 1: return A}$
 $\text{return (sort([x for x in A[1:] if x \leq A[0]]) + [A[0]] +}$
 $\text{sort([x for x in A[1:] if x > A[0]])}$

Synthesizer

Stanford

7

def sort(A):
 $\text{if len(A) == 4: return list(range(4))}$
 $\text{if len(A) == 3: return [1, 2, 4]}$
 return [9]

How about:

Stanford

Program Synthesis as Pragmatic Communication:

Ambiguity in Natural Languages:

- Try to determine what "they" means in below examples? -- easy for humans but can be difficult for a program

Ambiguity in natural language

- Human languages are extremely ambiguous, but that's typically not a problem for communication
- In fact, ambiguity is not a bug, it's a feature (of efficiency) [Piantadosi et al, 2012]
- Example: Winograd Schema Challenge
 - "The city councilmen refused the demonstrators a permit because **they feared** violence.
 - "The city councilmen refused the demonstrators a permit because **they advocated** violence."
- How do we do it?

Stanford

14

Pragmatic Reasoning:

- We (listener) select the smiley in the middle when finding a "friend with glasses", i.e. given a context C (I.e. hat, glasses, not hat/glasses) and utterance U (my friend has glasses)

Pragmatic Reasoning

- Human communication depends on cooperativity: we assume our conversational partner is trying to be as informative as possible
- We can use that in context to perform *pragmatic reasoning*
- Rational Speech Act (RSA) is a Bayesian model of how we choose (speaker) or interpret (listener) an utterance u given context c by recursively reasoning about the other party

- Rational Speech Act: Given a utterance u a Listener assigns roughly uniform probability to the choices he has been given, e.g.

Rational Speech Acts

- Assume these 3 objects and the following space of utterances: {"blue", "green", "circle", "square"}
- Given an utterance u , in RSA, a *literal listener* L_0 would assign: $P_{L_0}(o/u) \propto [[u]](o) P(o)$
- A *pragmatic speaker* S_1 that wants to refer to o chooses u reasoning about L_0 , balancing (1) how likely L_0 is to infer o given u , and (2) how costly is u (e.g., length) $P_{S_1}(u | s) \propto \exp(\alpha(\log P_{L_0}(o | u) - \text{Cost}(u)))$
- A *pragmatic listener* L_1 interprets utterance u assuming S_1 is speaking: $P_{L_1}(o | u) = P_{S_1}(u | o) P(o)$

- Using Pragmatic communication for program synthesis
 - Assuming finite set of programs and examples that can be given to the synthesizer
 - Works well with finite sets of programs and examples.

Program Synthesis as Pragmatic Communication

- Assuming a finite set of specifications and of programs, we can build a *meaning matrix*: $M[s][p] = 1$ if program p satisfies s

examples

program

Stanford

[Pu et al., 2020]

Large Language models can generate Code

- Idea is to use language modelling for code generation.
- LLMs are capable of predicting the next token given the context, so what if we train the GPT type of architectures on code database
- The initial attempt was made by Github copilot. So What did the dataset look like?
 - Code snippets including comments, docstrings and additional natural language
- The initial efforts was to train a 12B LLM with same architecture of GPT3 and it succeeded

Evaluating the LLM Code generation

Evaluating language models for code generation

- Synthesis challenge: given a Python docstring, generate the function implementation
- How to ensure problems were not seen during training?
- Authors introduced HumanEval, a manually created dataset of 164 problems
- Each problem has a set of hidden tests; a program is correct if it passes all hidden tests
- pass@k: probability that, out of k samples, at least one is correct

```
def incr_list(l):
    """Return list l with elements incremented by 1.
    Examples
    ======
    incr_list([2, 3, 4])
    [3, 4, 5]
    incr_list([1, 3, 5, 2, 4, 3, 9, 8, 12])
    [2, 4, 6, 3, 4, 4, 10, 12]
    """
    return [i + 1 for i in l]
```

Stanford

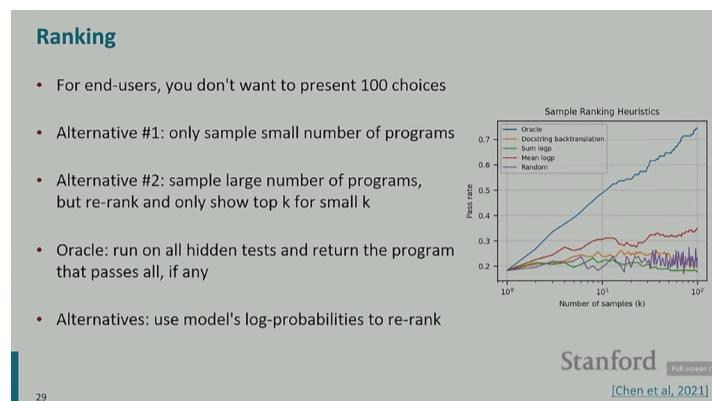
- Sampling – is generating N programs and selecting the one which passes the tests

Sampling vs Temperature

- Sampling more programs increases the chance of getting one right
- Trade-off between temperature and $P(\text{correct})$:
 - Low temperature: high likelihood (higher $P(\text{correct})$) less diversity
 - Higher temperature: lower likelihoods more diversity

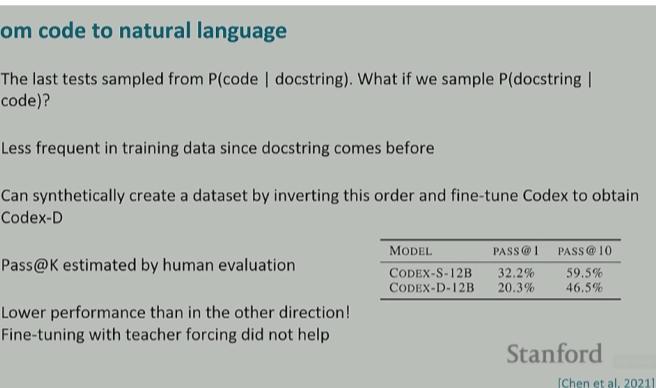


- Ranking – We can sample N programs but can't show user N programs as output.
Hence we need some selection criteria

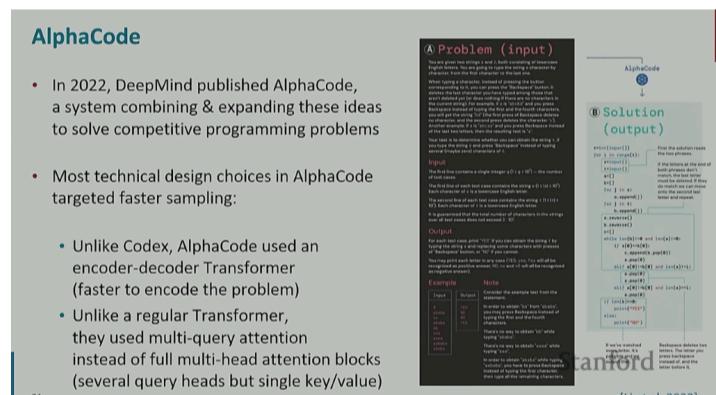


From Code to Natural Language

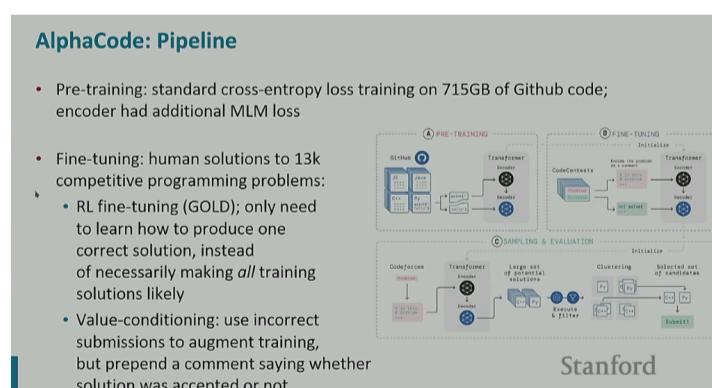
- What if we ask model to generate natural language given the code?
- Codex model was fine-tuned for such a task and results were worse as compared to $P(\text{Code} \mid \text{Natural Language})$



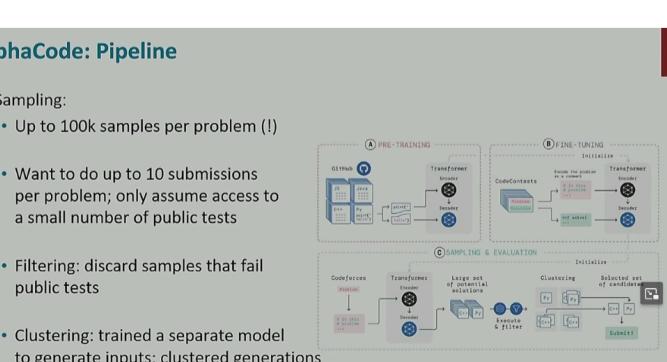
AlphaCode



- Changes from Codex:
 - Used encoder-decoder architecture instead of decoder only
 - Shared key-value heads and only query heads were different across layers
 - Addition of Masked language modelling loss
 - Usage of RL to learn how to produce only one correct solution
 - Inclusion of wrong outputs (with a comment that this output was wrong) in the training – called as value-conditioning



- Sampling: Unlike Codex they sampled 100k samples per problem!!!!
- Trained another model to generate inputs for the program to be tested, Grouped the programs based on their outputs



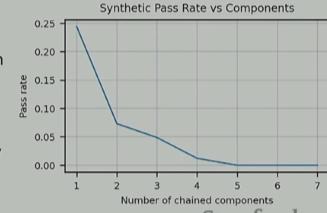
- Simple, if you generate-sample 10x more then you gotta to get few X factor better performance

A note on Compositionality

- Chaining of two simple problems creates difficulties for LLMs even when those problems are solvable for LLMs independently

Back to Codex: A note on compositionality

- If a human can trivially solve problem X (e.g., reverse a string), and also problem Y (e.g., compute string length), the problem "do X then Y" is still trivial
- This is not necessarily the case for LMs
- Codex paper reported an experiment with synthetic tasks made by chaining simple components like example above
- Result: performance decays exponentially as components increase, even if individually they're still trivial



[Chen et al., 2021]

36

Takeaway:

Code language models: takeaways

- Transformer models trained on large amounts of code have non-trivial performance in generating programs in real-world programming languages
 - These results were unimaginable just a few years ago
- Sampling, testing and filtering can get quite far
 - Although it gets expensive fast: *Training and evaluating our largest 41B model on Codeforces required a total of 2149 petaflop/s-days and 175 megawatt-hours [*16 times the average American household's yearly energy consumption (2019).]*
- Still, many of these experiments assume a setting that fundamentally differs from real-world programming
 - Well-defined, self-contained, short problems; extensive existing correctness & performance tests; only need standard libraries; ...

Stanford

37

Limitations and Discussion:

Limitations and Discussion

- Public code repositories have lots of code with bugs
- Generated code often has functional or security bugs. Still need to understand it!
- [Perry & Srivastava et al., 2022] ran a user study where participants solved programming tasks with and without Codex
 - "Overall, we find that participants who had access to an AI assistant based on OpenAI's codex-davinci-002 model wrote significantly less secure code than those without access"
 - "Additionally, participants with access to an AI assistant were more likely to believe they wrote secure code than those without access to the AI assistant."
- General psychological phenomenon known as Automation Bias

Stanford

