Program Synthesis using Neural Sequence Generation

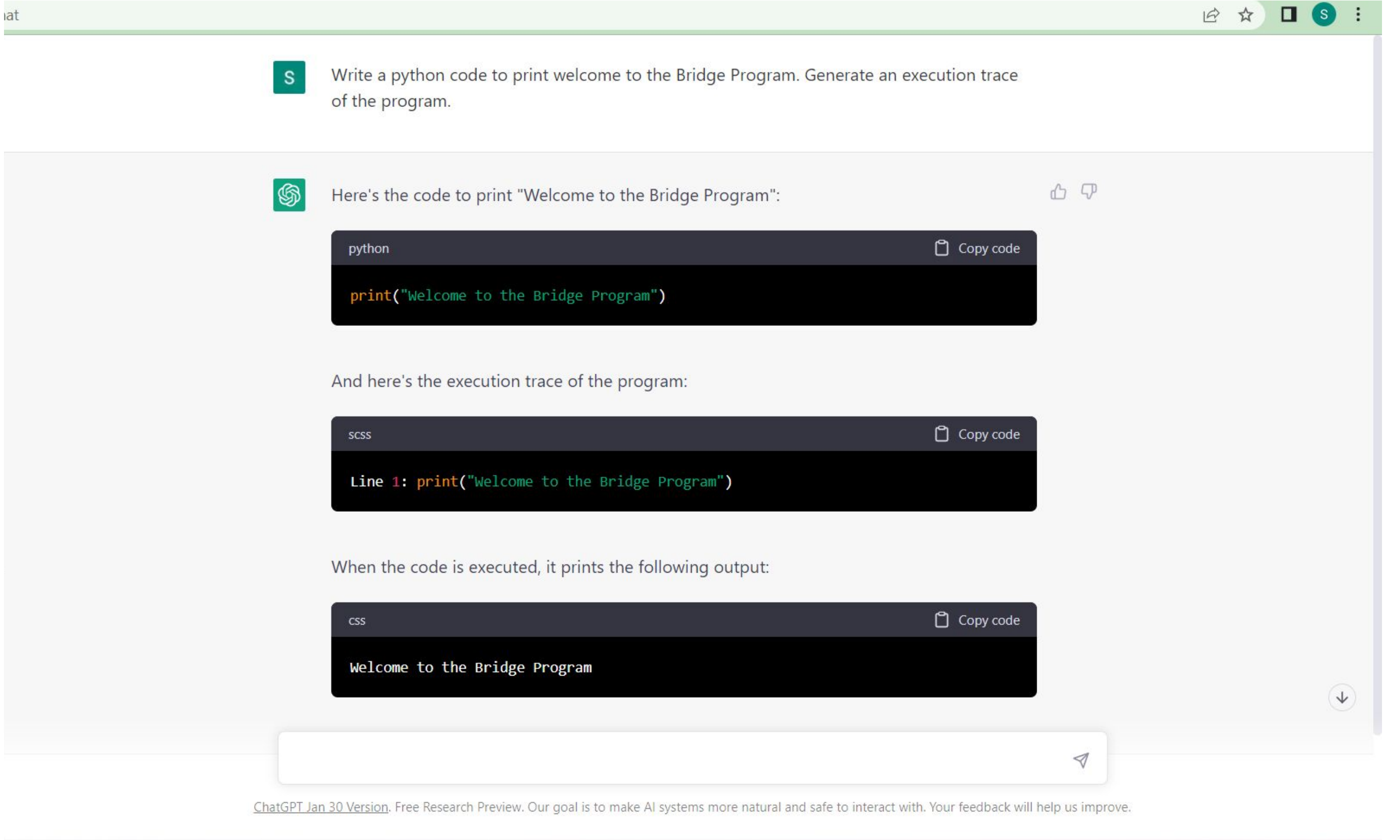
Shirish Karande and Ganesh Prasath Ramani,
Tata Consultancy Services - Research

Talk Sketch

- ❏ What is Program Synthesis ?
- ❏ Program Synthesis as Sequence Generation Task (with Karel as example)
- ❏ Pre-trained Language Models for complex code related tasks
- ❏ LLMs for code, tricks that help when fine-tuning is not an option
- ❏ Anecdotes about ChatGPT

What is Program Synthesis ?

A Transformer model generating python code from natural language



What is Program Synthesis ?

Generate code (in some language) from suitable specifications (intent)

Natural Language

Input-Output Examples

Code Smells and Properties

Traces/Demonstration

Another Formal Language

Python, C, C++, ...

SQL, SPARQL ....

Custom Language:

FlashFill, Karel, ...

What is Program Synthesis ?

DSL for String Modification

$$
\begin{array}{rcl}
\text{Program } p & := & \texttt{Concat}(\texttt{e}_1, \texttt{e}_2, \texttt{e}_3, ...) \\
\text{Expression } e & := & f \mid n \mid n_1(n_2) \mid n(f) \mid \texttt{ConstStr(c)} \\
\text{Substring } f & := & \texttt{SubStr}(\texttt{k}_1, \texttt{k}_2) \\
 & \mid & \texttt{GetSpan}(\texttt{r}_1, \texttt{i}_1, \texttt{y}_1, \texttt{r}_2, \texttt{i}_2, \texttt{y}_2) \\
\text{Nesting n} & := & \texttt{GetToken(t,i)} \mid \texttt{ToCase(s)} \\
 & \mid & \texttt{Replace}(\delta_1, \delta_2) \mid \texttt{Trim()} \\
 & \mid & \texttt{GetUpto(r)} \mid \texttt{GetFrom(r)} \\
 & \mid & \texttt{GetFirst(t,i)} \mid \texttt{GetAll(t)} \\
\text{Regex } r & := & t_1 \mid \cdots \mid t_n \mid \delta_1 \mid \cdots \mid \delta_m \\
\text{Type } t & := & \texttt{Number} \mid \texttt{Word} \mid \texttt{Alphanum} \\
 & \mid & \texttt{AllCaps} \mid \texttt{PropCase} \mid \texttt{Lower} \\
 & \mid & \texttt{Digit} \mid \texttt{Char} \\
\text{Case } s & := & \texttt{Proper} \mid \texttt{AllCaps} \mid \texttt{Lower} \\
\text{Position } k & := & -100, -99, ..., 1, 2, ..., 100 \\
\text{Index } i & := & -5, -4, -3, -2, 1, 2, 3, 4, 5 \\
\text{Character } c & := & \texttt{A}-\texttt{Z}, \texttt{a}-\texttt{z}, 0-9, !?, @... \\
\text{Delimiter } \delta & := & \&, .?!@()[]\%\{\}/ :; \$\#\text{"'} \\
\text{Boundary } y & := & \texttt{Start} \mid \texttt{End}
\end{array}
$$

Production Rules,
Terminal/Non-terminal symbols.

Specification in I/O examples

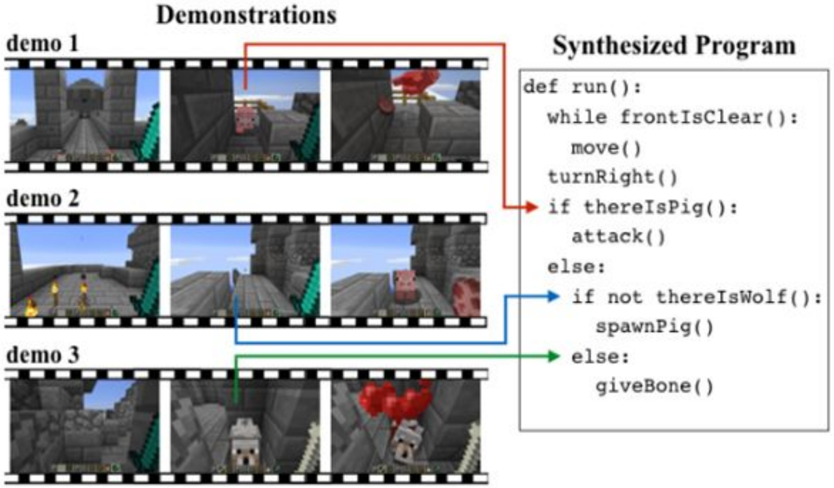
$I_1 =$ January	$O_1 =$ jan
$I_2 =$ February	$O_2 =$ feb
$I_3 =$ March	$O_3 =$ mar

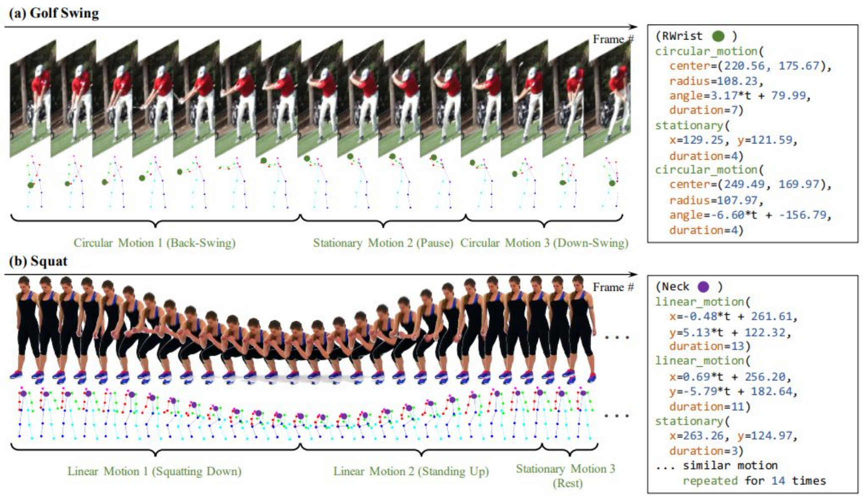
Program

$P =$ `ToCase(Lower, SubStr(1,3))`

RobustFill: Neural Program Learning under Noisy I/O. Devlin et. al., 2017

What is Program Synthesis ?

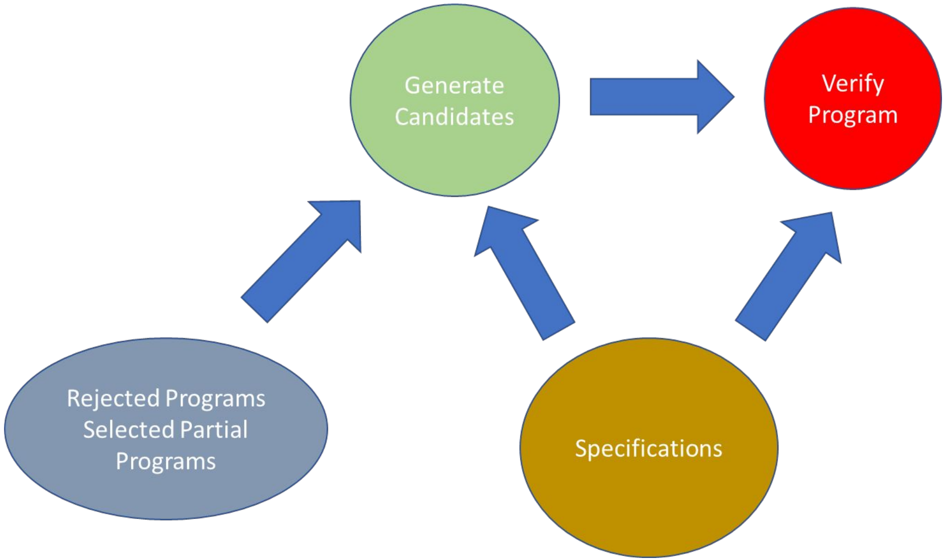




Neural Program Synthesis from Diverse Demonstration Videos, Sun et. al, 2018

Hierarchical Motion Understanding with Motion Programs, Kulal et. al, 2021

Explicit Enumeration and Search to Synthesize Programs



Program Space Representation

- Grammar to generate candidates
- Top Down v/s Bottom Up

Fast checking of equivalence

Fast checking for incorrectness

Neural Networks for Program Synthesis

How can use of Neural Networks help ?

(+1)	(-1)	(*2)	(/2)	(*-1)	(**2)	(*3)	(/3)	(*4)	(/4)	(>0)	(>0)	(%2==1)	(%2==0)	HEAD	LAST	MAP
.0	.0	.1	.0	.0	.0	.0	.0	1.0	.0	.0	1.0	.0	.2	.0	.0	1.0

FILTER	SORT	REVERSE	TAKE	DROP	ACCESS	ZIPWITH	SCANL1	+	-	*	MIN	MAX	COUNT	MINIMUM	MAXIMUM	SUM
1.0	1.0	.7	.0	.1	.0	.4	.0	.0	.1	.0	.2	.1	.0	.0	.0	.0

Figure 2: Neural network predicts the probability of each function appearing in the source code.

Deepcoder, Balog et. al, ICLR 2017.:

- ❑ "Sort and add" enumeration: Choose top k functions, feed to a traditional solver like Sketch or λ^2
- ❑ Depth First Search: When growing programs choose functions in an order guided by NN

*Distribution on any attributes can be learned.

Represent Search Space

- Statistical Distribution of Program

Represent Intent and Tasks

- Natural Language, Images,

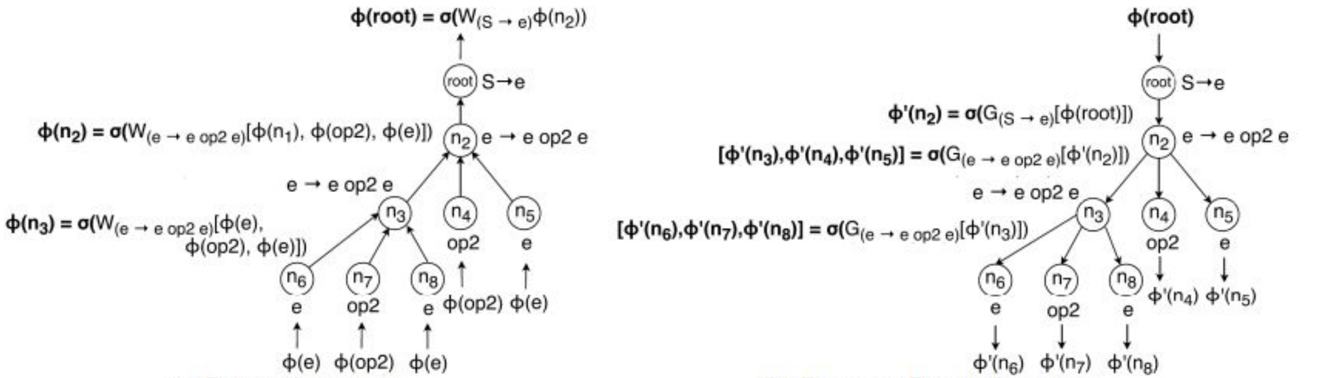
Represent Programs

- Predict Properties
- Predict Execution
- Predict Correctness (Repair)
- Predict Equivalence

Program Synthesis with Tree Structured Neural Architecture

- ❏ Representation for every symbol and production rule
- ❏ Forward and Reverse NN for each production rule
- ❏ Partial Program Trees of Depth T
- ❏ Task representation (I/O pairs) is concatenated at leafs

$$
\begin{array}{rcl}
\text{String } e & := & \text{Concat}(f_1, \cdots, f_n) \\
\text{Substring } f & := & \text{ConstStr}(s) \\
 & | & \text{SubStr}(v, p_l, p_r) \\
\text{Position } p & := & (r, k, \text{Dir}) \\
 & | & \text{ConstPos}(k) \\
\text{Direction Dir} & := & \text{Start} \mid \text{End} \\
\text{Regex } r & := & s \mid T_1 \cdots \mid T_n
\end{array}
$$



(a) Recursive pass

(b) Reverse-Recursive pass

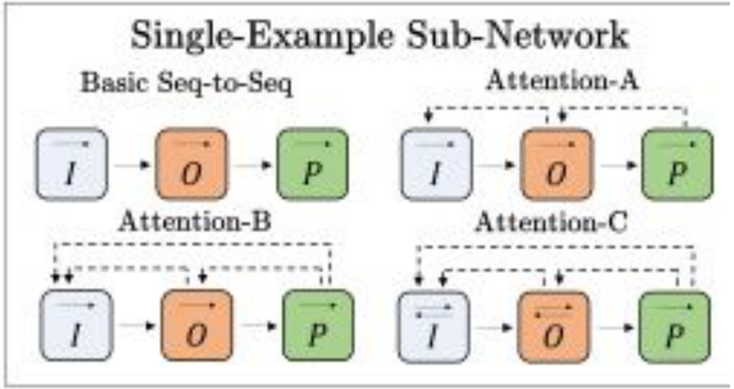
Neuro-Symbolic Program Synthesis, Parisotto et. al, 2017

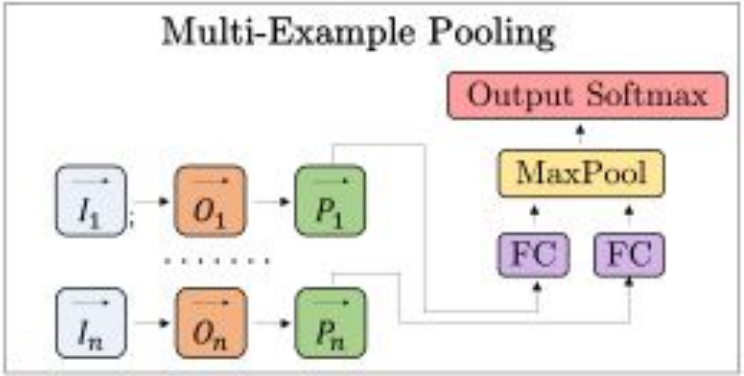
Program Synthesis as Sequence Generation

LSTM for representation of Input, Output, Program Generation

- 95 ASCII tokens for I and O
- 495 Program tokens for P
- Late Fusion: Predict program tokens after Max Pool
- Attention and Bi-Directional plays an important role

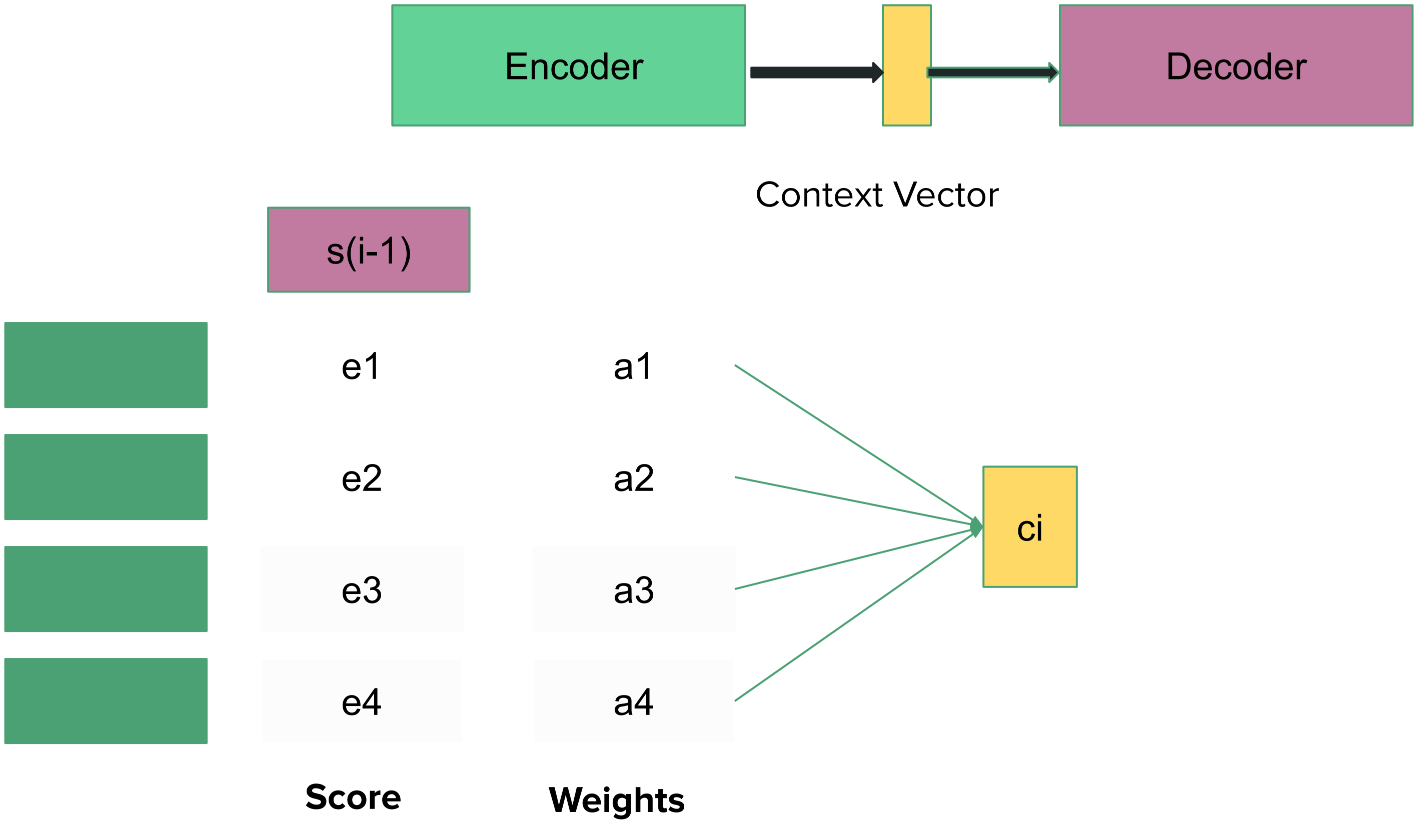
System	Beam	
	100	1000
Parisotto et al. (2017)	23%	34%
Basic Seq-to-Seq	51%	56%
Attention-C	83%	86%
Attention-C-DP	89%	92%





RobustFill: Neural Program Learning under Noisy I/O, Devlin et. al, 2017

Primer on Attention in sequence-sequence models



Neural machine translation by jointly learning to align and translate, Bahdanau et. al, 2017

Program Synthesis for Karel using Sequence Generation

Program Synthesis for Karel

$$
\begin{array}{rcl}
\text{Prog } p & := & \texttt{def main():}s \\
\text{Stmt } s & := & \texttt{while}(b):s \mid \texttt{repeat}(r):s \mid s_1;s_2 \\
 & \mid & a \mid \texttt{if}(b):s \mid \texttt{if}(b):s_1\texttt{else}:s_2 \\
\text{Cond } b & := & \texttt{markersPresent()} \mid \texttt{leftIsClear()} \\
 & \mid & \texttt{rightIsClear()} \mid \texttt{frontIsClear()} \mid \texttt{not}(b) \\
\text{Action } a & := & \texttt{move()} \mid \texttt{turnLeft()} \mid \texttt{turnRight()} \\
 & \mid & \texttt{pickMarker()} \mid \texttt{putMarker()} \\
\text{Cste } r & := & 0 \mid 1 \mid \cdots \mid 19
\end{array}
$$

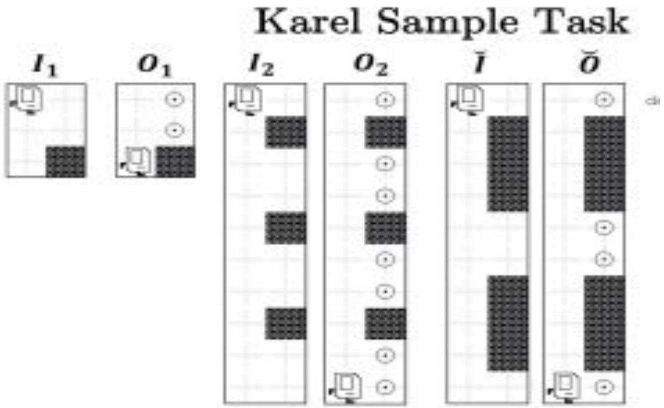
Scope Representation in Sequence

Conditionals: c(<body> c)

If: i(<body> i)

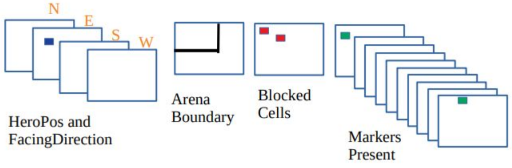
Else: e(<body> e)

Repeat: r(and r)



Neural Program Meta-Induction, Devlin et. al, 2017

- Feature Map = (16,18,18)



State : {
HeroPos : (2,2);
HeroDir : 1;
Boundary : { (0,0), (10, 10) };
BlockedCells : { (1,2), (4,4) };
Markers : {(2,1) = 1,
(1,5) = 10 }
}

(a) Grid representation

(b) State Representation

Figure 3 : Example of Karel world

Channel (1:4) : Hero Position and Facing Direction
Channel (5) : Arena Boundary (Hero has to move inside the arena)
Channel (6) : Cells with walls (where Hero cannot go)
Channel (7:16) : Number of markers at a position.
Value of any marker in Channel 7 = 1
...
Value of any marker in Channel 16 = 10

Program Synthesis for Karel

1. CNN to embed I/O example
2. Late Fusion as done in RobustFill
3. Task Embedding + Previous Token

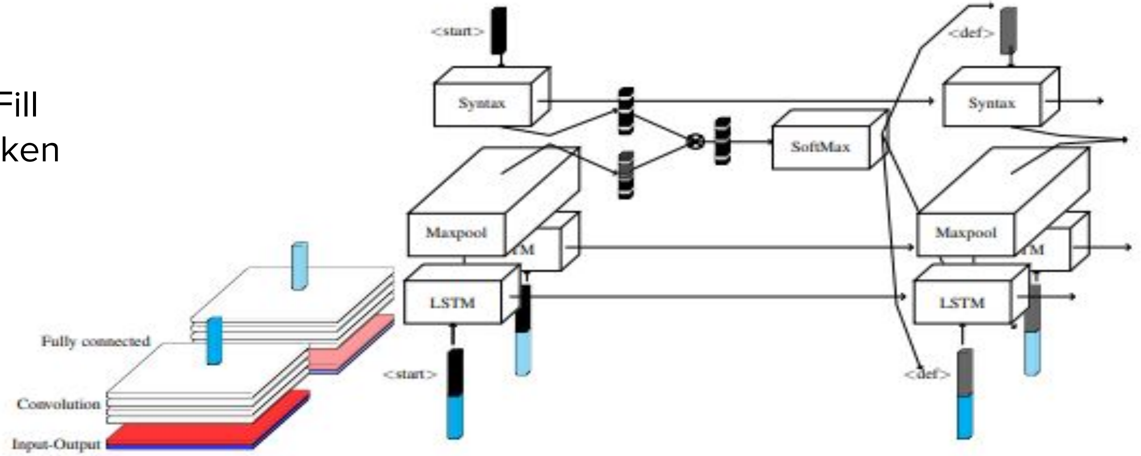
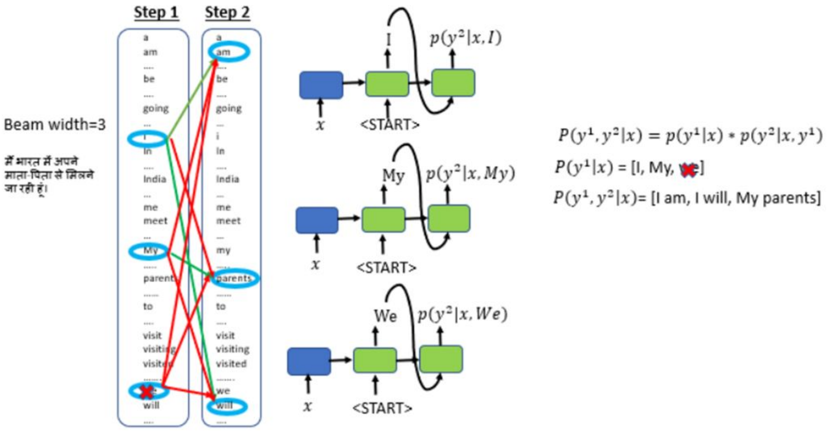


Figure 2: Architecture of our model. Each pair of Input-Output is embedded jointly by a CNN. One decoder LSTM is run for each example, getting fed in a concatenation of the previous token and the IO pair embedding (constant across timestep). Results of all the decoders are maxpooled and the prediction is modulated by the mask generated by the syntax model. The probability over the next token is then obtained by a Softmax transformation.

Leveraging Grammer and Reinforcement Learning For Neural Program Synthesis, Devlin et. al, 2017

Grammar constrained decoding

Beam Search



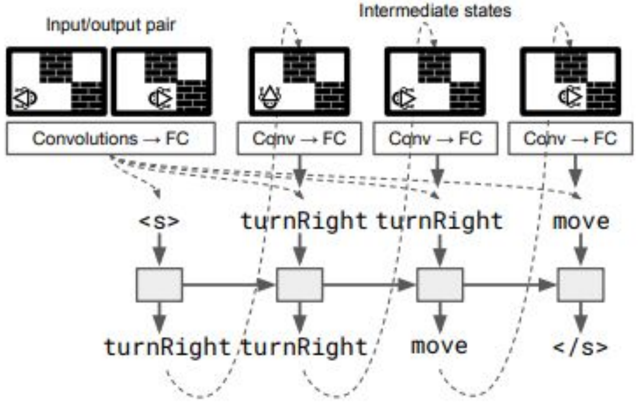
https://towardsdatascience.com/an-intuitive-explanation-of-beam-search-9b1d744e7a0f

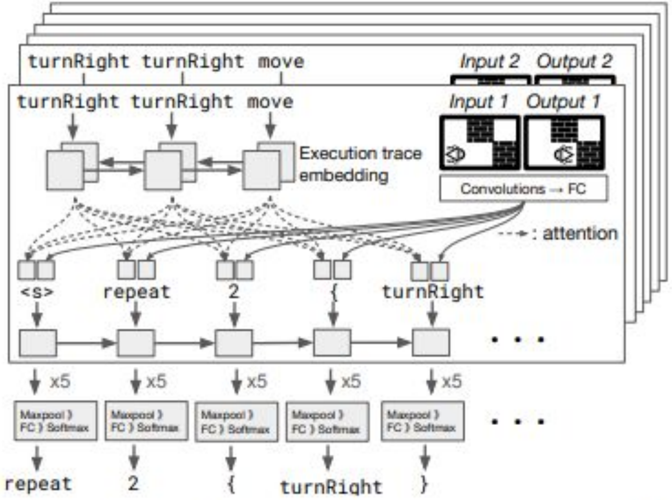
Grammatical correctness can be used to apply masks on the tokens that can be selected at each step.

Leveraging Grammar and Reinforcement Learning For Neural Program Synthesis, Devlin et. al, 2017

Neural generation of traces for program synthesis

Synthesize traces from the Input-Output States
Use Karel interpreter to replay actions and find out corresponding intermediate states.
Use Bi-LSTM to get an embedding of trace, state interleaved,
Continue using late fusion

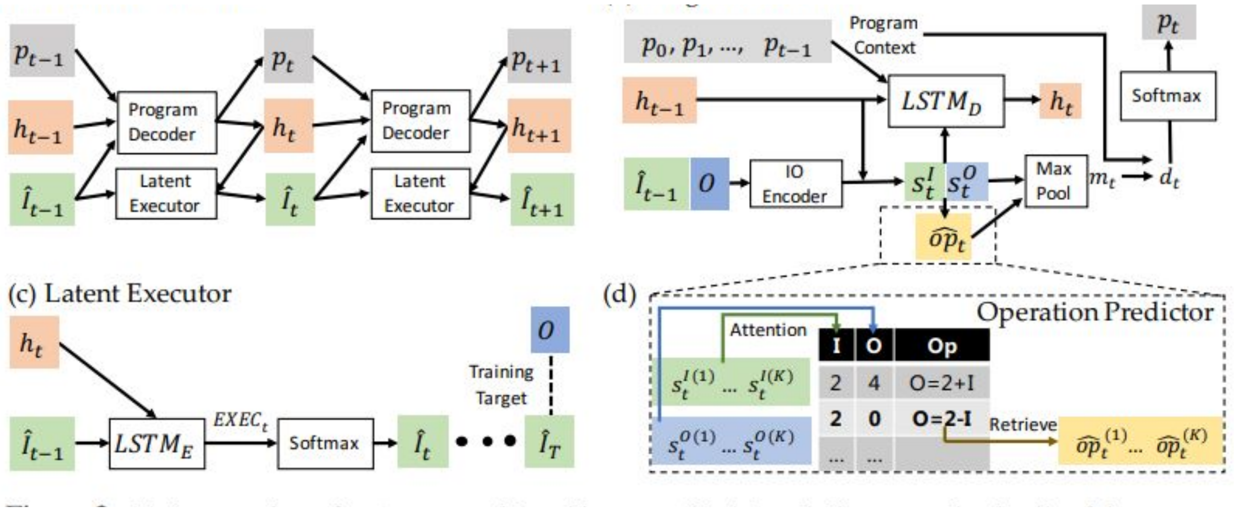




Improving Neural Program Synthesis with Inferred Execution Traces, Shin et. al, 2018

Latent Execution for Neural Program Synthesis

- LSTM to represent Intermediate State s.t. If given as input to remaining partial program it leads to same output.
- Use updated states for synthesis (work for snippets of C as well)

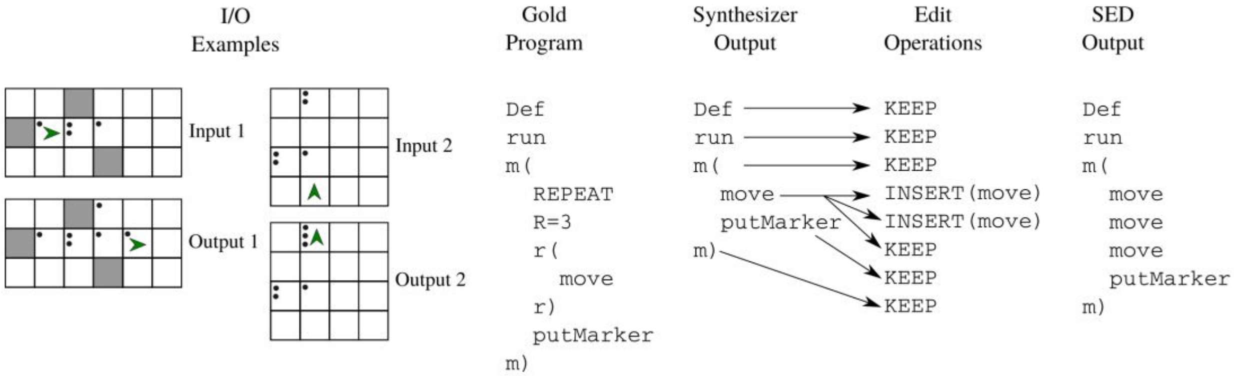


Latent Execution for Neural Program Synthesis, Chen et. al, 2021

Neural Debugger

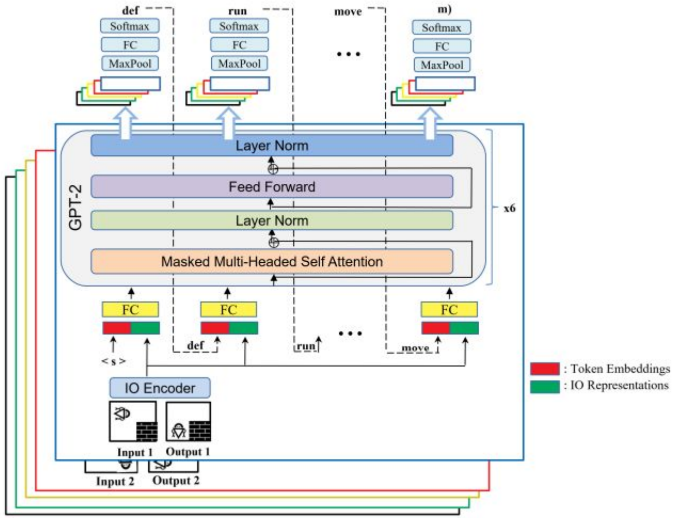
Train an LSTM to debug the program generated by the synthesizer LSTM

Debugging Language: KEEP, DELETE, INSERT, REPLACE



Synthesize, Execute and Debug: Learning to Repair for Neural Program Synthesis, Gupta et. al, 2021

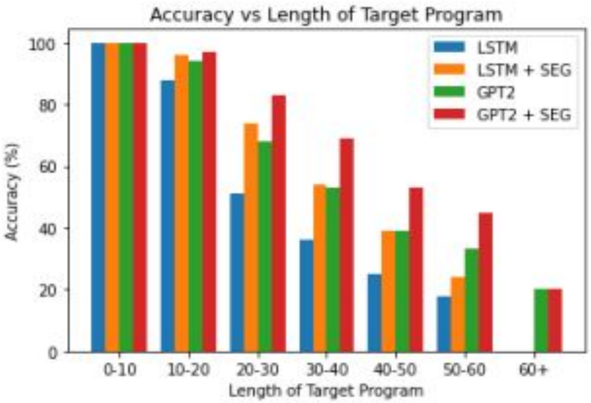
Transformers for the Synthesizer



Transformer Decoder for Synthesizer

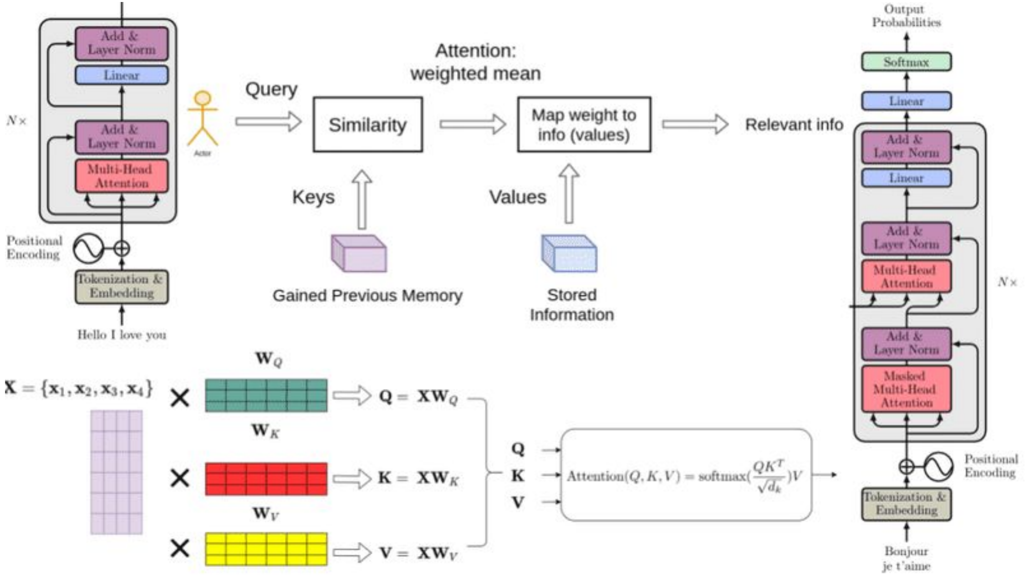
Performance gain greater on longer programs

Method	Top-1 Gen	Exact Match
Leveraging Grammar + LSTM [3]	73.67%	39.94%
IO->Trace->Program [32]	81.30%	42.80%
Latent Execution [7]	83.68%	41.12%
Execution Guided Decoding [6]	86.04%	40.88%
Learning to Repair [16]	89.80%	43.48%
Execution Guided Decoding (Ensemble) [6]	92.00%	47.08%
LSTM Decoder	73.48%	40.88%
LSTM + SEG	84.48%	43.28%
Transformer Decoder	82.40%	43.36%
Transformer + SEG	89.64%	44.80%
Transformer + Debugger	**90.44%**	**44.88%**



Are Transformers All That Karel Needs, Garg et. al, 2021

Why Transformers?



Source: https://theaisummer.com/transformer/

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

- Infinite Reference
- Non Sequential Processing
- Parallelism (multi GPU)
- Less Complex Computations

Attention is all you need, Vaswani et. al. (2017)

Code Generation using Pre-trained (Fine-tunable) Language Models

Tasks

Category	Task	Dataset Name	Language	Train/Dev/Test Size	Baselines	Task definition
Code-Code	Clone Detection	BigCloneBench	Java	900K/416K/416K	CodeBERT	Predict semantic equivalence for a pair of codes.
		POJ-104	C/C++	32K/8K/12K		Retrieve semantically similar codes.
	Defect Detection	Devign	C	21k/2.7k/2.7k		Identify whether a function is vulnerable.
	Cloze Test	CT-all	Python, Java, PHP, JavaScript, Ruby, Go	-/-/176k		Tokens to be predicted come from the entire vocab.
		CT-max/min	Python, Java, PHP, JavaScript, Ruby, Go	-/-/2.6k		Tokens to be predicted come from {max, min}.
	Code Completion	PY150	Python	100k/5k/50k	CodeGPT	Predict following tokens given contexts of codes.
		GitHub Java Corpus	Java	13k/7k/8k		
	Code Repair	Bugs2Fix	Java	98K/12K/12K	Encoder-Decoder	Automatically refine codes by fixing bugs.
	Code Translation	CodeTrans	Java-C#	10K/0.5K/1K		Translate the codes from one programming language to another programming language.
Text-Code	NL Code Search	CodeSearchNet, AdvTest	Python	251K/9.6K/19K	CodeBERT	Given a natural language query as input, find semantically similar codes.
		CodeSearchNet, WebQueryTest	Python	251K/9.6K/1k		Given a pair of natural language and code, predict whether they are relevant or not.
	Text-to-Code Generation	CONCODE	Java	100K/2K/2K	CodeGPT	Given a natural language docstring/comment as input, generate a code.
Code-Text	Code Summarization	CodeSearchNet	Python, Java, PHP, JavaScript, Ruby, Go	908K/45K/53K	Encoder-Decoder	Given a code, generate its natural language docstring/comment.
Text-Text	Documentation Translation	Microsoft Docs	English-Latvian/Danish/Norwegian/Chinese	156K/4K/4K		Translate code documentation between human languages (e.g. En-Zh), intended to test low-resource multi-lingual translation.

CodeXGLUE, A machine learning benchmark dataset for code understanding and generation, Chen et. al, 2021

Metrics of Performance

- Exact Match
- Program is grammatically correct
- Program satisfies the I/O example used to specify intent and can generalize to held out examples as well.

Other notions of equivalence haven’t been used (atleast not consistently):

- Similar logic is being employed
- Similar usage of resources

Structured Information about the Code

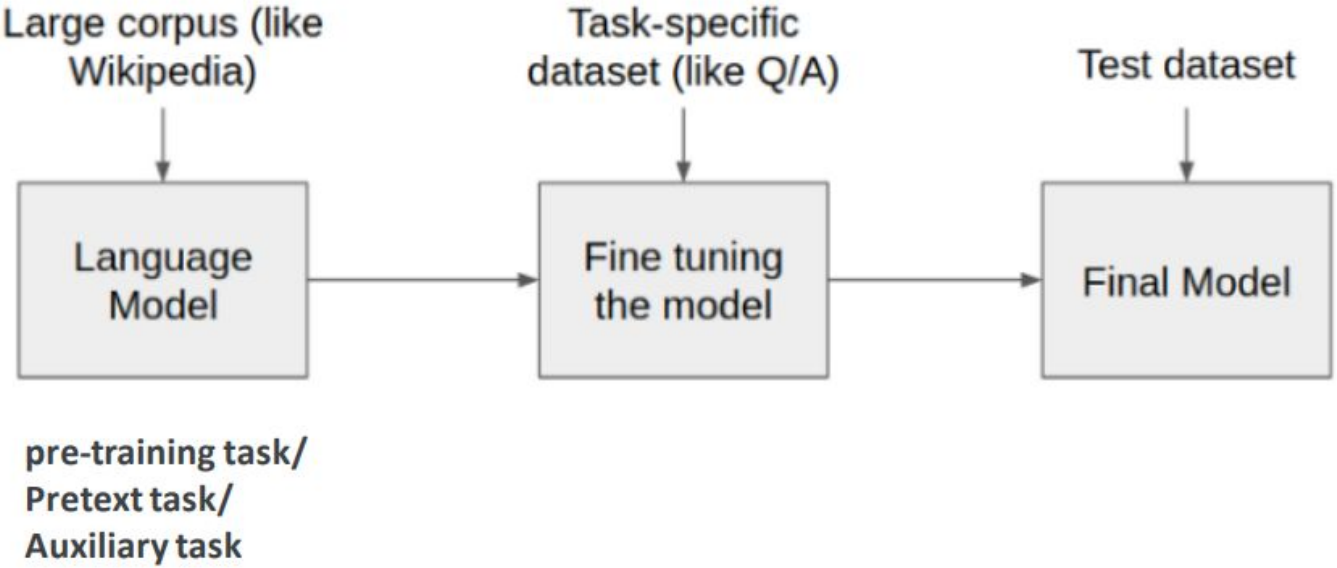
What:

- ❑ Abstract Syntax Tree
- ❑ Control flow
- ❑ Data flow

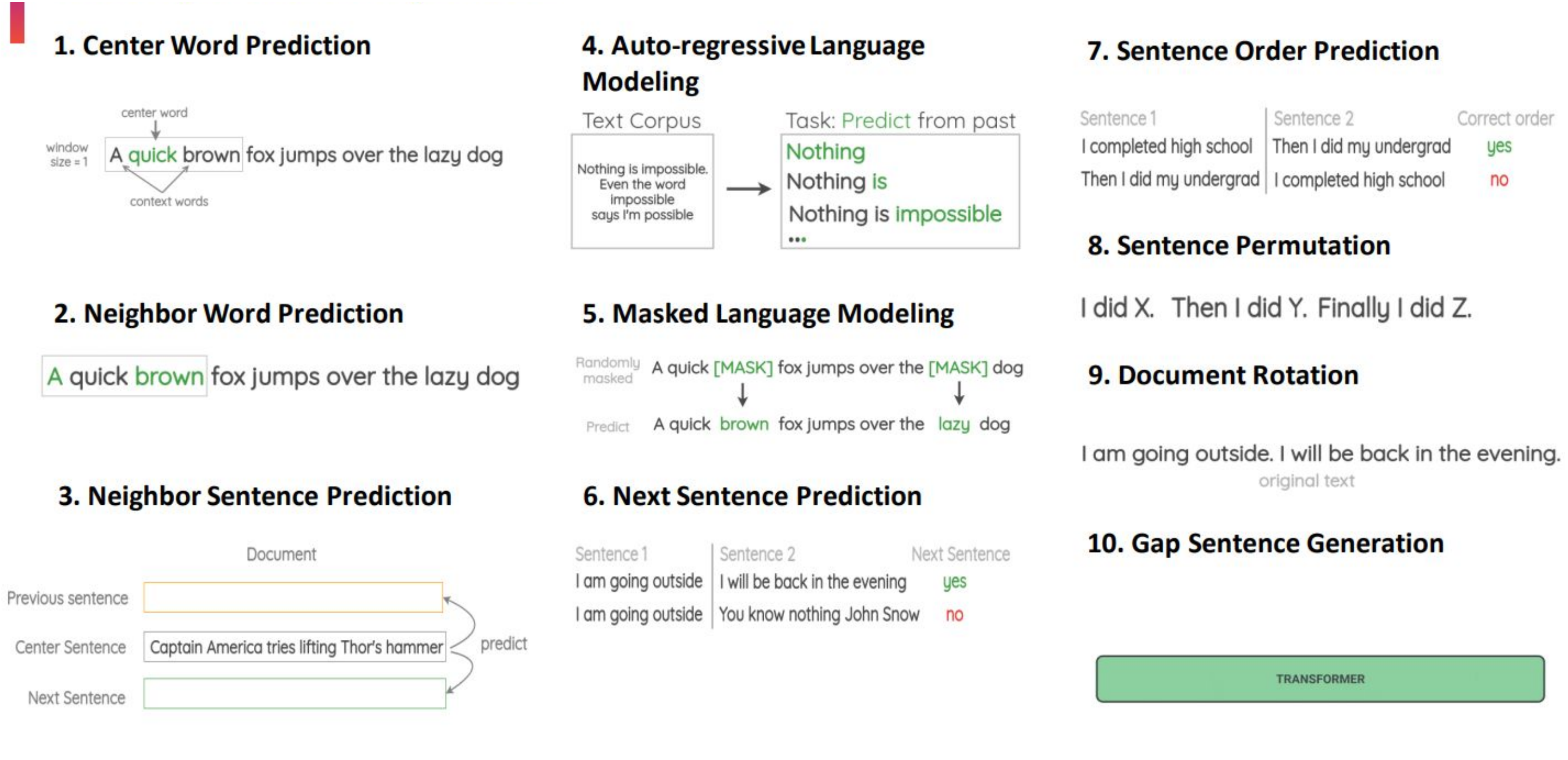
How:

- ❑ Linearize use preorder traversal, path decomposition, etc
- ❑ Structured pre-training tasks
- ❑ Overlaying the embedding space

Idea behind using pre-trained LMs



Self Supervision Tasks in NLP



https://amitness.com/2020/05/self-supervised-learning-nlp/

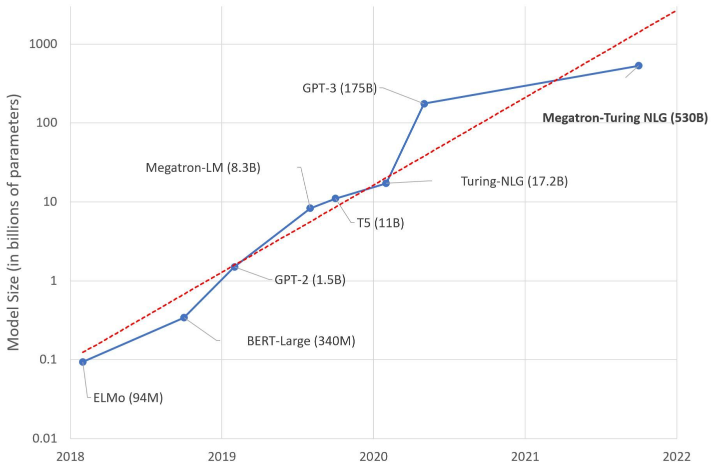
Self-Supervision tasks for pretraining code LMs

Code BERT	Encoder Only	MLM, Replaced Token Detection
Code GPT	Decode Only	Next token prediction
Transcoder	Encoder Only	Crosslingual MLM, Denoising AE, Back Translation
PL BART	Encoder - Decoder	MLM, Token Deletion, Token Infilling (masked span)
Code T5	Encoder - Decoder	Masked Span, Identifier Tagging, Identifier Masking, Dual Generation
CodeGen	Decoder only	Next Token Prediction

Code Generation using Pre-trained Large Language Models

Large Language Models

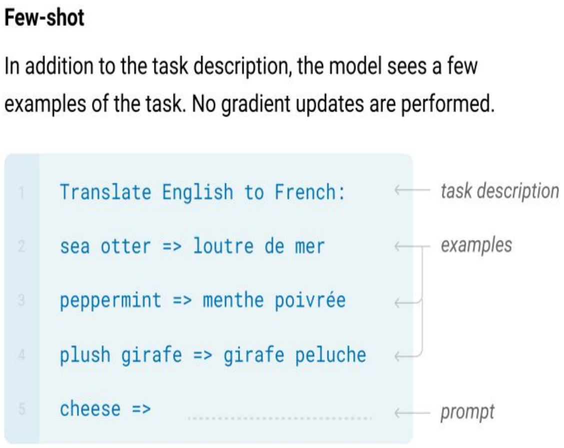
- Size usually in hundreds of billions of parameters
    - Lambda 135 billion, GPT3 - 175 billion, Megatron 530 billion
- Trained for multitasking, prompt/prefix based task selection
- Shows in Context learning and other emergent capabilities.



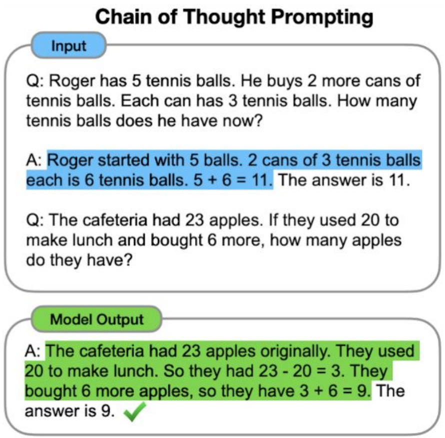
Challenges:

- Generally too large for fine tuning
- Model parameters not openly available
- Access to Model itself is restricted to the APIs

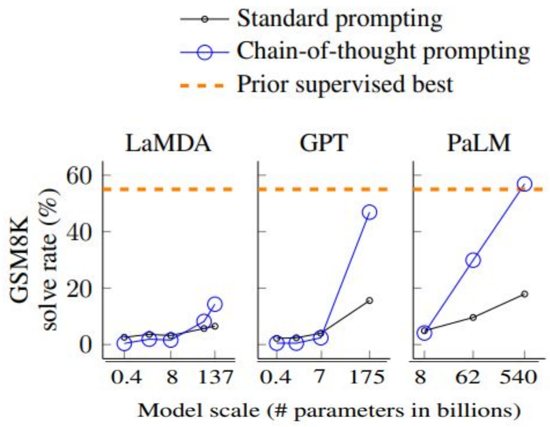
Non Invasive Generation improvements for LLMs



Comparable to PBE



Comparable to Execution traces



[Attention is all you need, Vaswani et.al (2017)](#)
[Chain of Thought prompting elicits reasoning in large language models, Wei et.al (2022)](#)

CoT Example - GPT3

129990102991 + 90 =
1300909901991

2390+8 = 2398
8978 + 5 = 8983
129990102991 + 90 = 129990103
071 + 10 = 081

2390+8 = 2398
8978 + 5 = 8983
129990102991 + 90 =
Lets think step by step:

1. Add the last digit (90) to the ones digit (1) of the number, which gives us 91.
2. Add the carried over digit (9) to the tens digit (9) of the number, which gives us 18.
3. Add the carried over digit (1) to the hundreds digit (0) of the number, which gives us 1.
4. Add the carried over digit (0) to the thousands digit (2) of the number, which gives us 2.
5. Add the carried over digit (0) to the ten thousands digit (9) of the number, which gives us 9.
6. Add the carried over digit (0) to the hundred thousands digit (9) of the number, which gives us 9.
7. Add the carried over digit (0) to the millions digit (1) of the number, which gives us 1.

Therefore, the answer is 129990103081.

Can Large Language models do zero shot coding?

Generate a python function to return thrice the input with five added:

```
def thrice_plus_five(input):
 return (input*3)+5
```

Generate a Karel program to move 5 cells and take a left turn

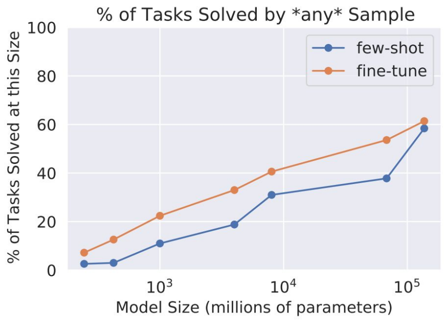
```
//Karel program to move 5 cells and take a left turn

move();
move();
move();
move();
move();
turnLeft();
```

Generate a Karel program to move 5 cells and take a left turn. Shorten the program as much as possible:

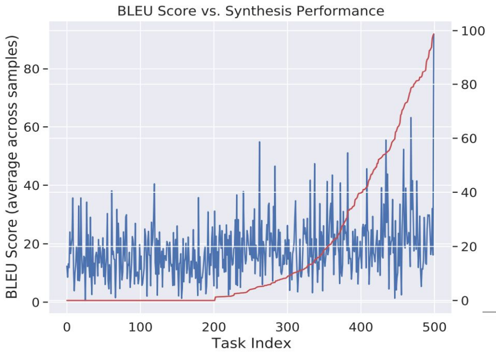
```
move();
left();
move(
left();
move();
left();
move();
left();
move();
left();
```

```
'''
Generate a karel program using python API to move N steps and turn left.
'''

from karel.stanfordkarel import *

def main():
  n = int(input('Enter the number of steps: '))

  for i in range(n):
    move()

  turn_left()

if __name__ == '__main__':
  run_karel_program()
```

Performance of LLMs in Code Generation



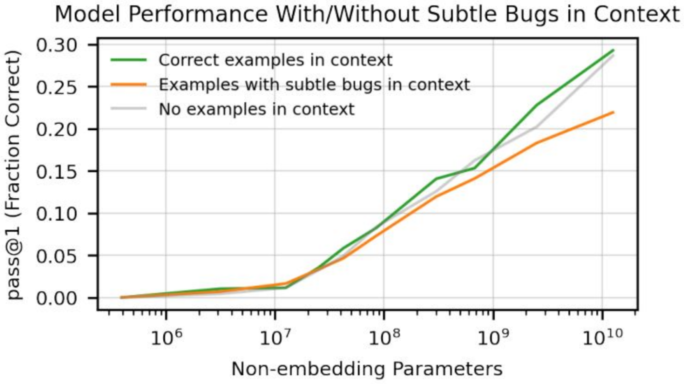
Percentage of samples solved increases with model size and shows no sign of flattening.



Text generation metrics such as Bleu score correlates weakly with program performance.

Factors improving performance

- Model Size
- Prompt Quality
- Few-shot example relevance
- Human - Model alignment
- Sampling Strategy / Post Processing



Codex

- Evaluated on MBPP & MathQA datasets
- Program synthesis tasks has to be approached and evaluated differently than text generation

Program synthesis using Large Language models, Austin. Et.al (2021)
Evaluating Large Language Models Trained on Code, Chen Et. al (2021)

Reliable code generation using LLMs

What can we control?

- Relevance to the task
- Context length
- Few shot selection strategy
- Decomposing and planning through Sketching.

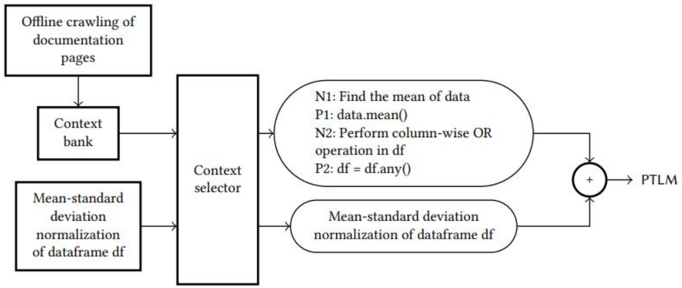
- Sampling Strategies
- Selection and scoring strategies
- Correction / Post Processing

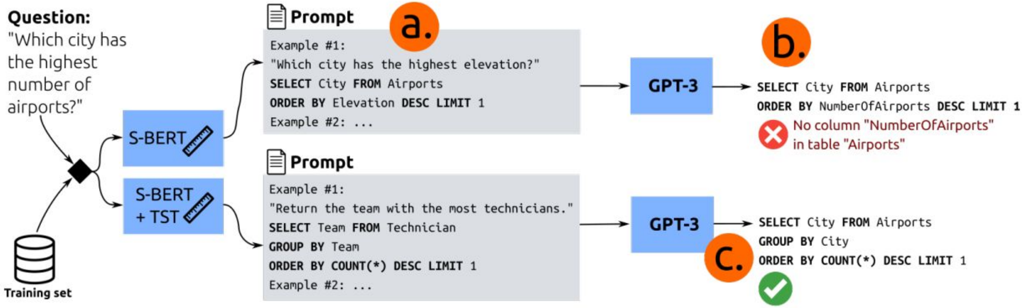
Context Specification

Example Prompt from an API reference:

Load a feather-format object from the file path.
pandas.read_feather(*path, columns=None, use_threads=True, storage_options=None*)

- Prompt Repository
    - Public API Documentations
    - Code Summaries
    - Few shot examples
    - Private Libraries
    - Schema Information





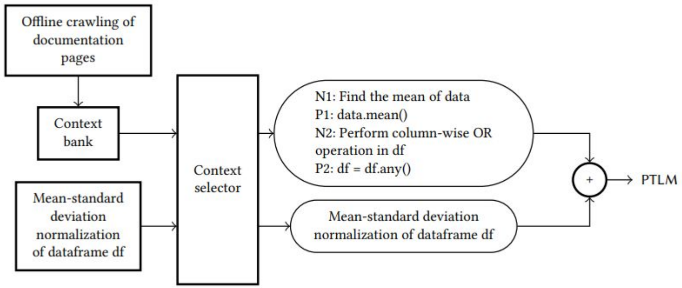
TST - Target Similarity Tuning, CSD - Constrained semantic Decoding

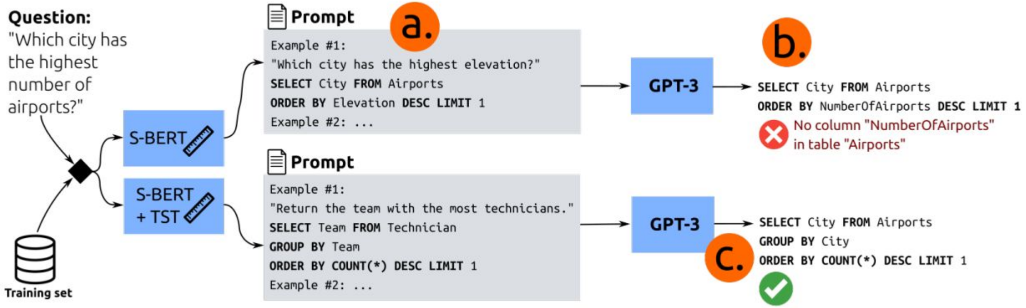
[JigSaw: Large Language models meet program synthesis, Jain et.al (2021)]

[Synchromesh: Reliable Code Generation From Pre-Trained Language Models, Poesia et.al, 2022]

Context Selection

- Context selection using Similarity with Query
  - TF-IDF
  - Embedding distance
  - TST - AST Edit distance of Programs



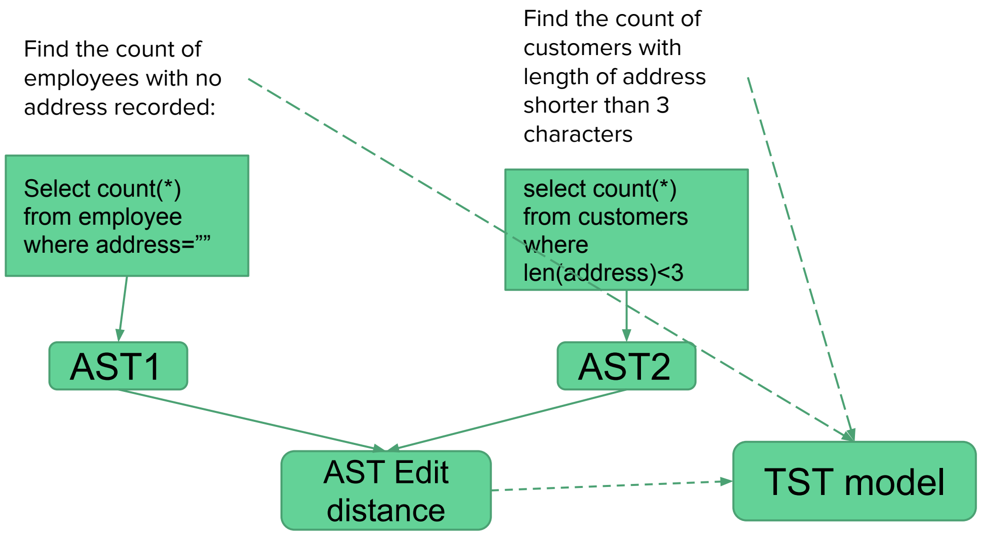


TST - Target Similarity Tuning, CSD - Constrained semantic Decoding

[JigSaw: Large Language models meet program synthesis, Jain et.al (2021)]

[Synchromesh: Reliable Code Generation From Pre-Trained Language Models, Poesia et.al, 2022]

Target Specific Tuning

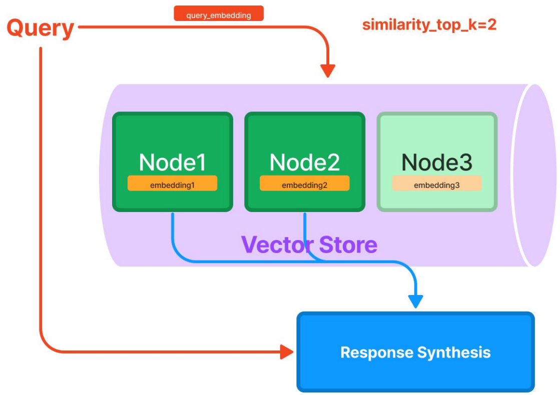


- Model learns from random pairs of positive and negative examples from training set.
- Natural language queries are the strings to match and the AST edit distance is the expected similarity score.
- Once trained we have a way to compare NL queries based on the target they achieve.

Handling Large context repositories

- Large documentations, API references, manuals etc. can be splitted into meaningful chunks and vectorized using embeddings from GPT3, BERT etc.
- Store the embeddings in Vector Databases
- Retrieve based on Query similarity.

Tools like GPT3-Index, Faiss, Weaviate can help.



Node: Corresponds to a chunk of text from a Document

Image from GPT3-Index

Output Sampling strategies

- Beam Search may not be the right choice
- Not every sample will be valid
- Grammar / Execution based scoring and selection can reduce errors

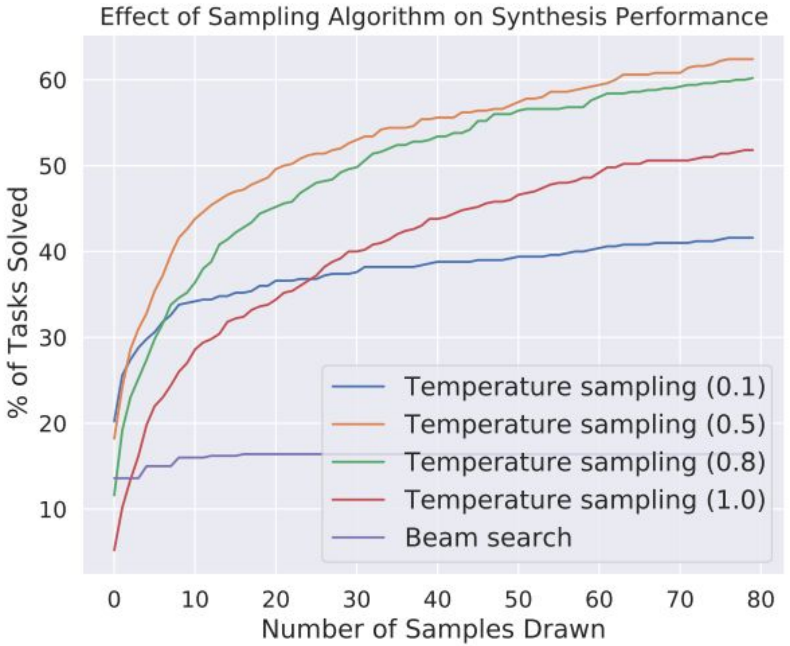


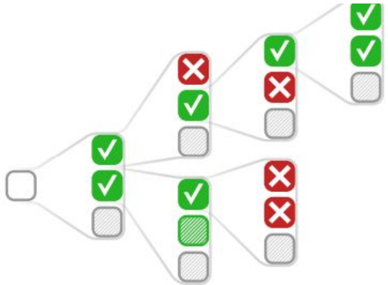
Figure 9: Higher temperatures achieve better scaling with more samples, but perform worse with a smaller budget.

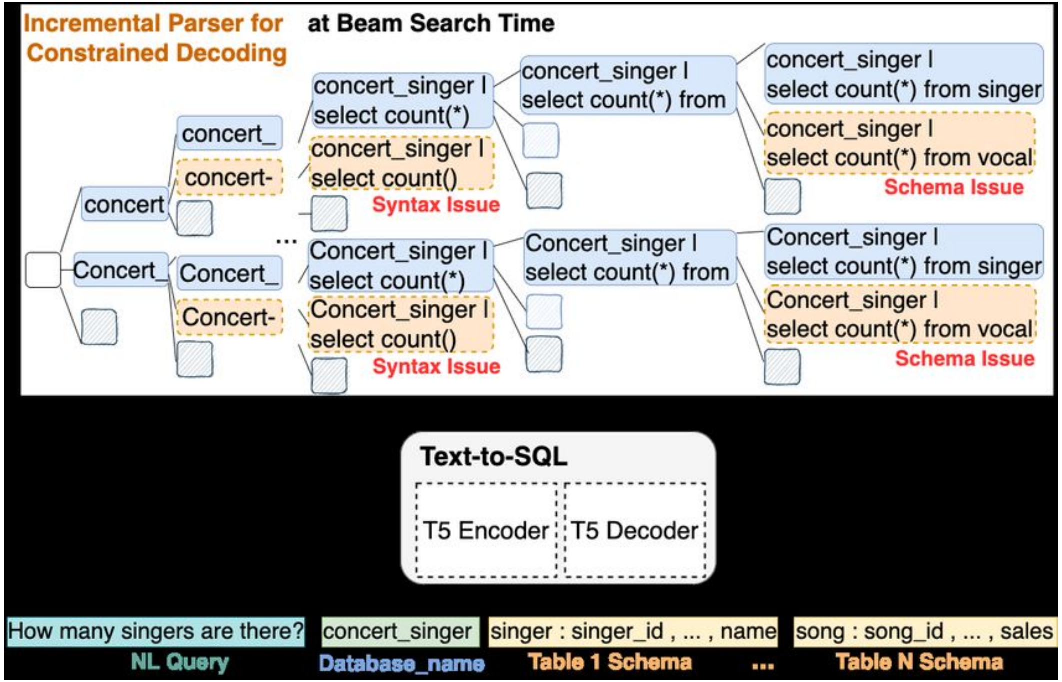
[Program synthesis using Large Language models, Austin. Et.al (2021)]

Grammar based Guidance

Task : NL → SQL (Spider, CoSQL)

- Constrain Autoregressive decoding through incremental parsing
- Filter output tokens based on SQL Grammar & Database Schema.
- Lexing, Parsing with and without Guards.

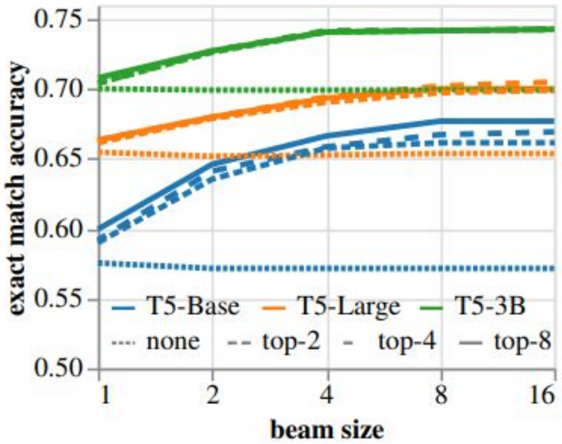




[PICARD: Parsing Incrementally for Constrained Auto Regressive Decoding from Language models, Scholak (2022)]()

Grammar Based Guidance

Performance similar to SOTA even with smaller models.



System	Development EM%	Development EX%	Test EM%	Test EX%
BRIDGE v2 + BERT (ensemble)† (Lin et al., 2020)	71.1	70.3	67.5	68.3
SMBoP + GRAPPA† (Rubin and Berant, 2021)	74.7	75.0	69.5	71.1
RATSQL + GAP† (Shi et al., 2021)	71.8	-	69.7	-
DT-Fixup SQL-SP + RoBERTa† (Xu et al., 2021)	75.0	-	70.9	-
LGESQL + ELECTRA† (Cao et al., 2021)	75.1	-	**72.0**	-
T5-Base (Shaw et al., 2021)	57.1	-	-	-
T5-3B (Shaw et al., 2021)	70.0	-	-	-
T5-Base (ours)	57.2	57.9	-	-
T5-Base+PICARD	65.8	68.4	-	-
T5-Large	65.3	67.2	-	-
T5-Large+PICARD	69.1	72.9	-	-
T5-3B (ours)	69.9	71.4	-	-
T5-3B+PICARD	74.1	76.3	-	-
T5-3B†	71.5	74.4	68.0	70.1
T5-3B+PICARD†	**75.5**	**79.3**	**71.9**	**75.1**

** A dagger (†) indicates use of database content, otherwise schema only.*

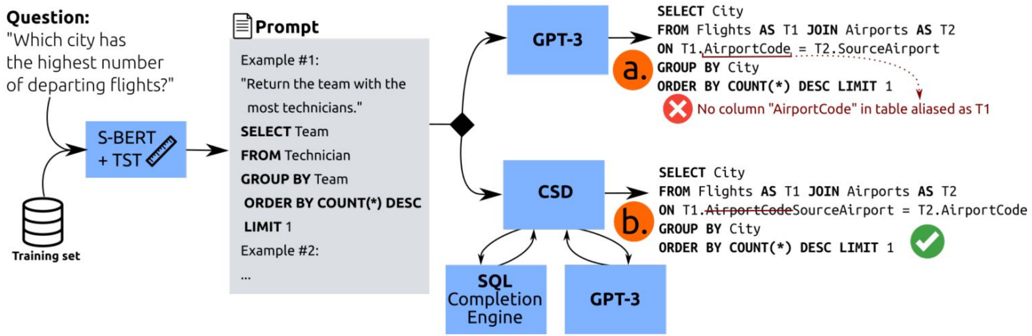
PICARD: Parsing Incrementally for Constrained Auto Regressive Decoding from Language models, Scholak (2022)

Output Guidance Strategies

TST - Target Similarity Tuning
CSD - Constrained semantic Decoding

Token by token decoding where the set of next tokens are given by a constraint engine.

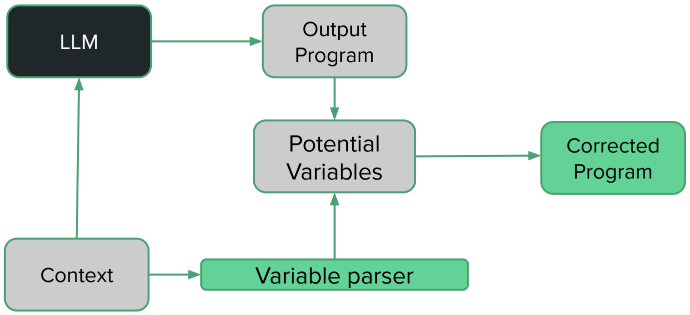
The constraint might be based on Syntax or Semantics of the target.



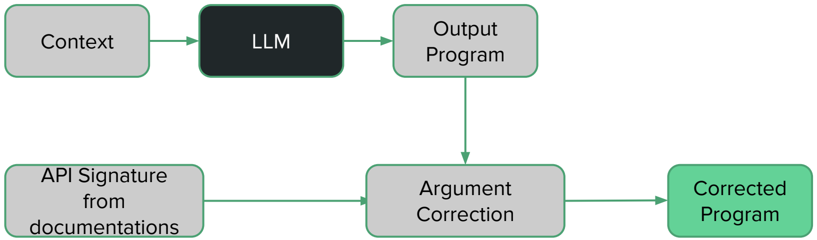
Language	Constraint	Example of partial program	Valid/Invalid Examples
SQL	A valid identifier must follow after `AS`.	`SELECT Name, Role FROM User AS` ^	U ✓ T1 ✓ 2 ✗
	Column names must come from schema, even behind aliases.	`SELECT U.Name FROM User AS U WHERE U.` ^	Name ✓ DoB ✓ Birthday ✗

Synchromesh: Reliable Code Generation From Pre-Trained Language Models, Poesia et.al, 2022

Output Correction methods - Rule Based



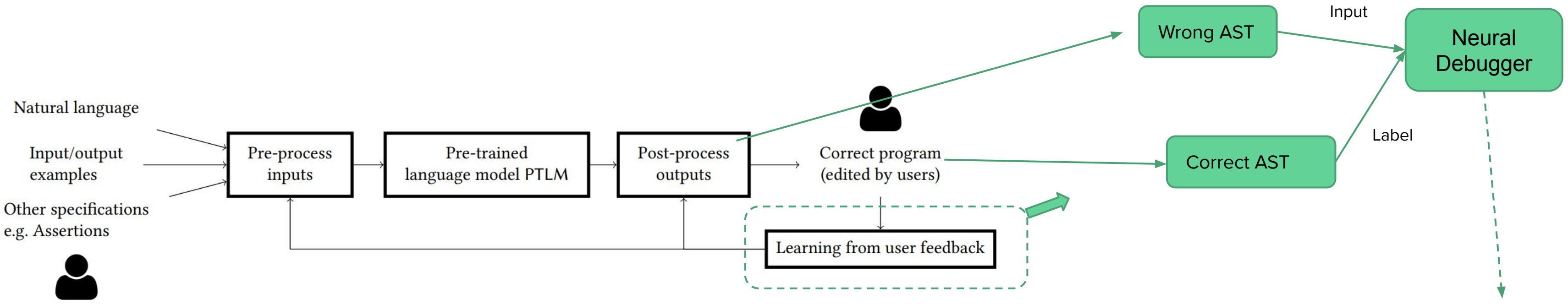
Variable Corrections



Argument Corrections

JigSaw: Large Language models meet program synthesis, Jain et.al (2021)

Output Corrections - Learnable Methods

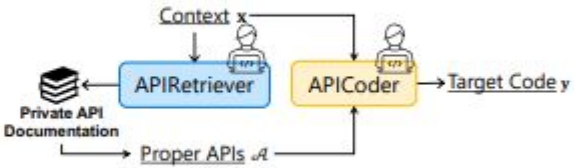


This eventually becomes a part of post processing improving the accuracy over time

[JigSaw: Large Language models meet program synthesis, Jain et.al (2021)]

Addressing Proprietary Libraries/APIs

LM models are mostly unaware of proprietary libraries/apis: Most enterprise projects will use proprietary libraries which may have never been exposed to the Models. This increases the chances of failure in retrieving the right program for the task.



APICoder	APIs	PandasEval pass@1	PandasEval pass@10	PandasEval pass@100
CodeGen	No API	14.24	30.71	46.04
	Perfect	11.21	33.59	48.47
	Top-2	9.54	29.02	40.56
CodeGenAPI	No API	13.58	34.95	46.51
	Perfect	19.96	42.36	53.43
	Top-2	11.25	28.61	39.48
		NumpyEval		
CodeGen	No API	19.31	40.89	60.58
	Perfect	21.41	41.08	56.38
	Top-2	18.30	35.12	48.46
CodeGenAPI	No API	16.55	29.48	42.52
	Perfect	24.83	41.47	54.41
	Top-2	12.67	27.32	35.62

Table 3: Results of CodeGen and CodeGenAPI on PandasEval and NumpyEval.

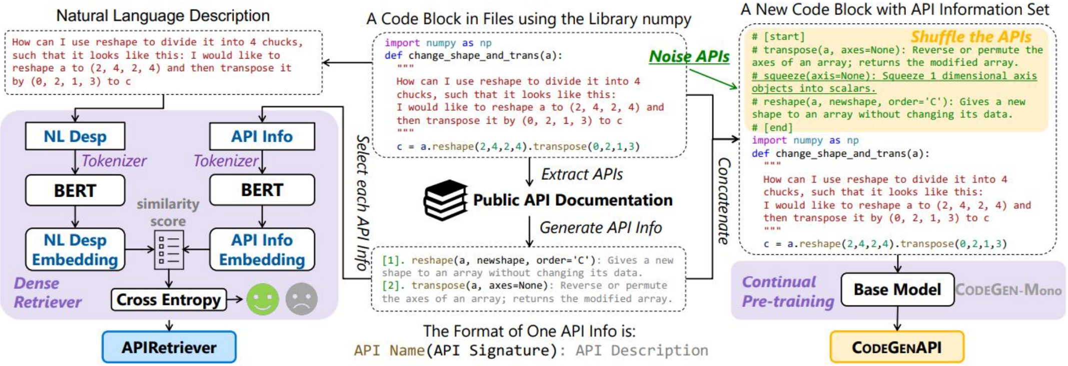


Figure 3: The training process of APIRetriever and CodeGenAPI.

Figures from the paper "When Language model meets Private Library"

When Language Model Meets Private Library Zan, et.al (2022)