Neural Program Synthesis

Nadezhda Chirkova

Centre of Deep Learning and Bayesian Methods
HSE University

NATIONAL RESEARCH UNIVERSITY

p(B|A)yesgroup.ru

Outline

- Problem statement, datasets, metrics
- Approaches
  - Supervised learning-based
  - Self-supervised pretraining-based
- Remaining challenges
- Codex: details, demos, experiments - next week with Sergey Troshin

Preliminaries — basic NLP: Seq2seq, BERT, BLEU, BPE

Outline

- $\boxed{\text{Problem statement, datasets, metrics}}$
- Approaches
  - Supervised learning-based
  - Self-supervised pretraining-based
- Remaining challenges
- Codex: details, demos, experiments - next week with Sergey Troshin

Preliminaries — basic NLP: Seq2seq, BERT, BLEU, BPE

Problem statement

Given a list of citations counts, where each citation is a nonnegative integer, **write a function h_index that outputs the h-index**. The h-index is the largest number h such that h papers have each least h citations.

→

```
def h_index(counts):
  n = len(counts)
  if n > 0:
    counts.sort()
    counts.reverse()
    h = 0
    while (h < n and
        counts[h]-1>=h):
      h += 1
    return h
  else:
    return 0
```

Exampe from https://arxiv.org/pdf/2105.09938.pdf

Problem statement

Given a list of citations counts, where each citation is a nonnegative integer, **write a function h_index that outputs the h-index**. The h-index is the largest number h such that h papers have each least h citations.

[3,0,6,1,4] → 3
[1,4,1,4,2,1,3,5,6] → 4
[1, 0] → 1
[1000,500,500,250,100,100,100,100,100,75,50,30,20,15,15,10,5,2,1] → 15

```
def h_index(counts):
  n = len(counts)
  if n > 0:
    counts.sort()
    counts.reverse()
    h = 0
    while (h < n and
          counts[h]-1>=h):
      h += 1
    return h
  else:
    return 0
```

(may be one type of input or both)

Exampe from https://arxiv.org/pdf/2105.09938.pdf

Problem statement: options

- *Program specification*: text, tests
- *Additional input*: tests, context (e. g. class definition or database)
- *Program language*: domain-specific language (DSL) or general-purpose language (e. g. Python or Java)
- *Needed knowledge*: basic programming, APIs, algorithms, …

Datasets

- *Program specification*: text, **tests**
- *Additional input*: tests, context (e. g. class definition or database)
- *Program language*: **domain-specific language (DSL)** or general-purpose language (e. g. Python or Java)
- *Needed knowledge*: **basic programming**, APIs, algorithms,

FlashFill dataset

Input v_1	*Output*
BTR KRNL WK CORN 15Z	*15Z*
CAMP DRY DBL NDL 3.6 OZ	*3.6 OZ*
CHORE BOY HD SC SPNG 1 PK	*1 PK*
FRENCH WORCESTERSHIRE 5 Z	*5 Z*
O F TOMATO PASTE 6 OZ	*6 OZ*

$$\downarrow$$

$$\mathtt{SubStr}(v_1, \mathtt{Pos}(\epsilon, \mathit{NumTok}, 1), \mathit{CPos}(-1))$$

Size: 205 examples

Source: random data generation for training

S. Gulwani. Automating string processing in spreadsheets using input-output examples. POPL'11

Datasets

- *Program specification*: **text**, tests
- *Additional input*: tests, **context** (e. g. class definition or database)
- *Program language*: domain-specific language (DSL) or **general-purpose language** (e. g. Python or Java)
- *Needed knowledge*: **basic programming**, **APIs**, algorithms, …

CONCODE dataset

```
public class SimpleVector implements Serializable {
  double[] vecElements;
  double[] weights;

  NL Query: Adds a scalar to this vector in place.
  Code to be generated automatically:
  public void add(final double arg0) {
    for (int i = 0; i < vecElements.length; i++){
      vecElements[i] += arg0;
    }
  }

  NL Query: Increment this vector
  Code to be generated automatically:
  public void inc() {
    this.add(1);
  }
}
```

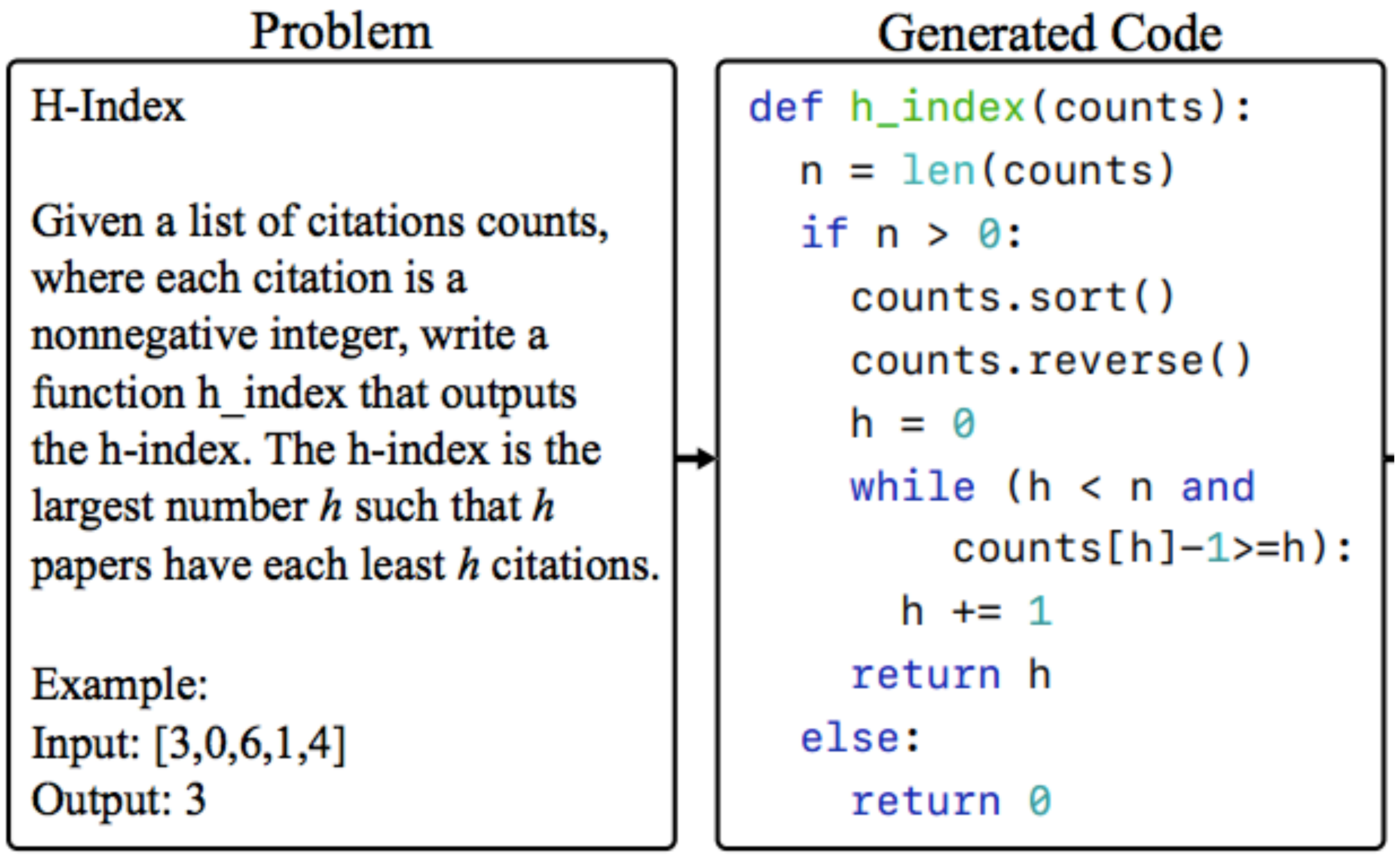
Size: 100K methods
Source: GitHub

Srinivasan Iyer et al.Mapping Language to Code in Programmatic Context. EMNLP 2018

Datasets

- *Program specification*: **text**, tests
- *Additional input*: **tests**, ~~context (e. g. class definition)~~ **self-contained**
- *Program language*: domain-specific language (DSL) or **general-purpose language** (e. g. Python or Java)
- *Needed knowledge*: **basic programming**, **APIs**, **algorithms**

APPS dataset



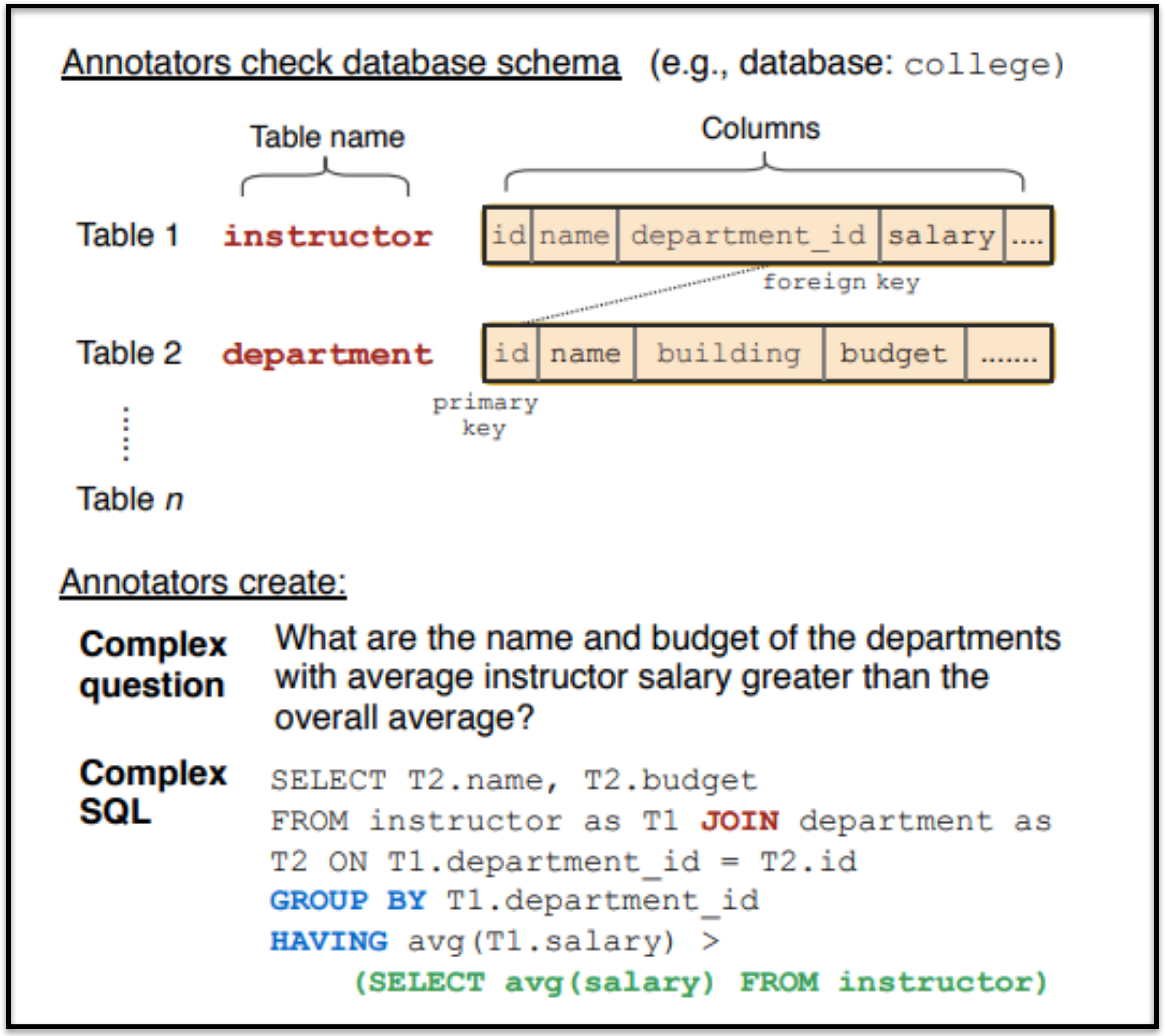
Size: 10K problems / 260K programs + *tests*!
Source: programming contests, e.g. Codeforces
3 levels of task difficulty

Dan Hendrycks et al. Measuring Coding Challenge Competence With APPS. arXiv 2021

Datasets

- *Program specification*: **text**, tests
- *Additional input*: tests, context (e. g. class definition or **database**)
- *Program language*: domain-specific language (DSL) or general-purpose language (e. g. Python or Java) **SQL**
- *Needed knowledge*: **basic programming**, APIs, algorithms,…

Spider dataset



Size: 10K questions over 200 databases

Source: 11 computer science students

Tao Yu et al. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. EMNLP 2018

Quality evaluation

- **Exact match (EM)**: generated program exactly equals to the ground-truth one
- **BLEU:** textual similarity between generated and ground-truth programs
- **CodeBLEU:** similarity at token-level, Abstract Syntax Tree level and data-flow level (Ren et al, arxiv 2020)
- **Executing code and running tests**

BLEU poorly correlates with functional correctness

Performance of GPT-Neo model on the APPS dataset:

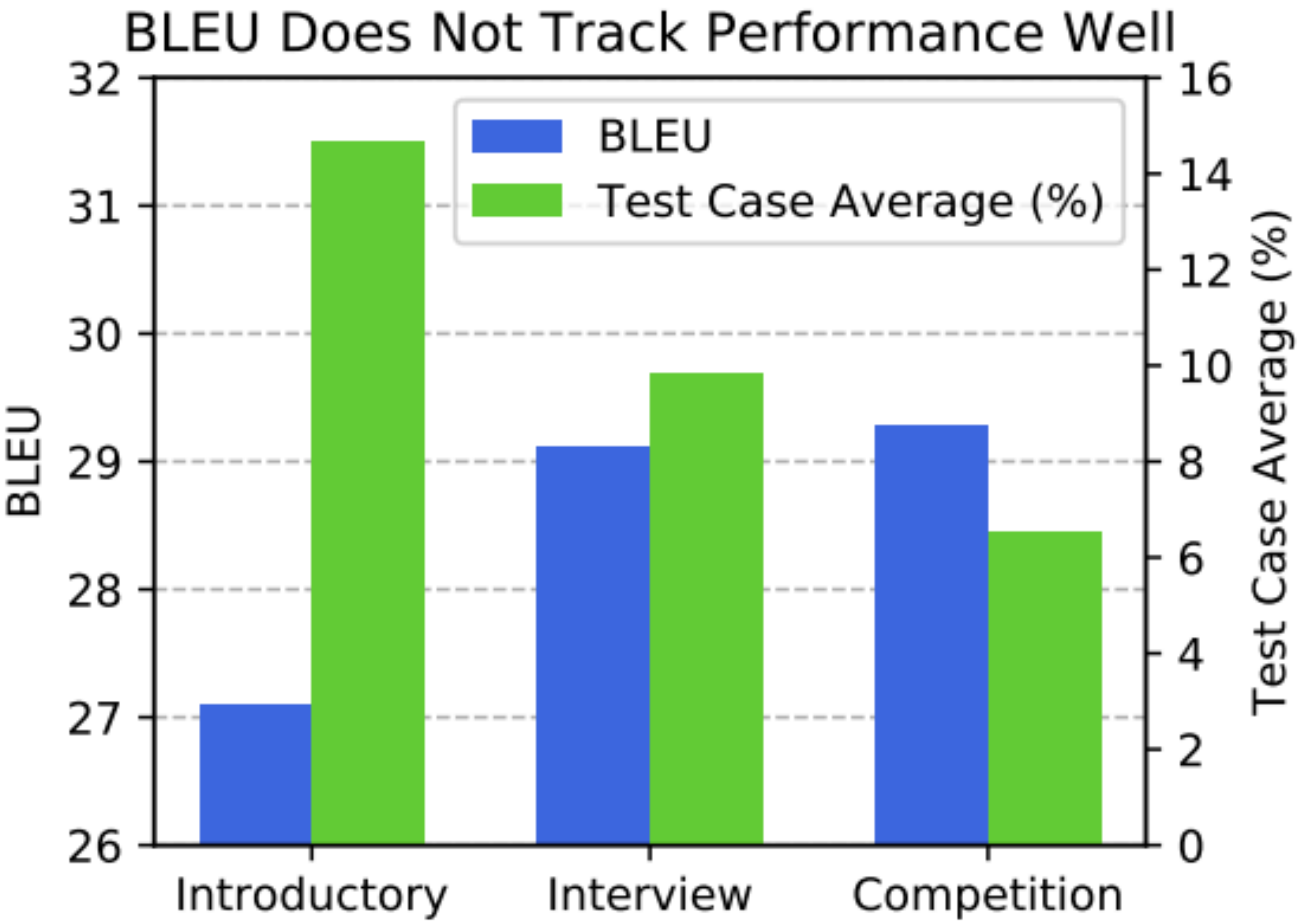
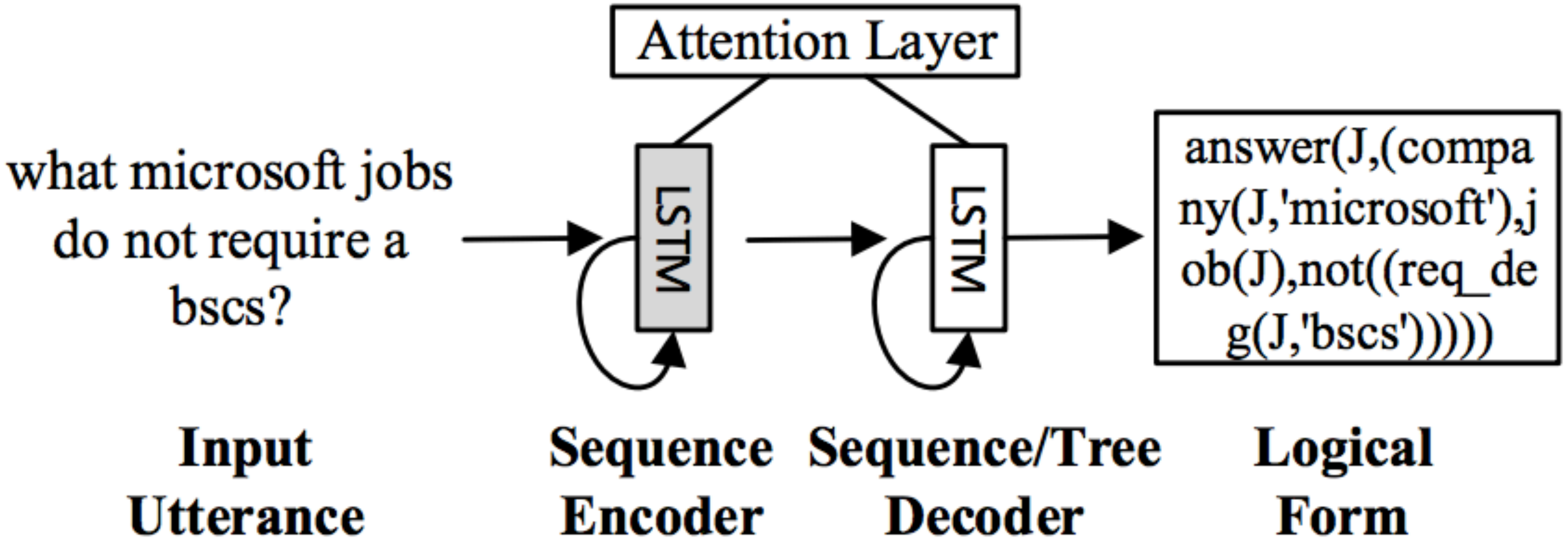


Image from Dan Hendrycks et al. Measuring Coding Challenge Competence With APPS. arXiv 2021

Outline

- Problem statement, datasets, metrics
- Approaches
  - Supervised learning-based
  - Self-supervised pretraining-based
- Remaining challenges
- Codex: details, demos, experiments - next week

Basic Seq2seq approach



- *Input*: text, O*utput*: code as token sequence or syntactic tree
- *Training* in a supervised manner or using reinforcement learning
- *Architectural details*: attention mechanism, copy mechanism
- *Add-ons*: grammar-based constraints, supervision from execution…

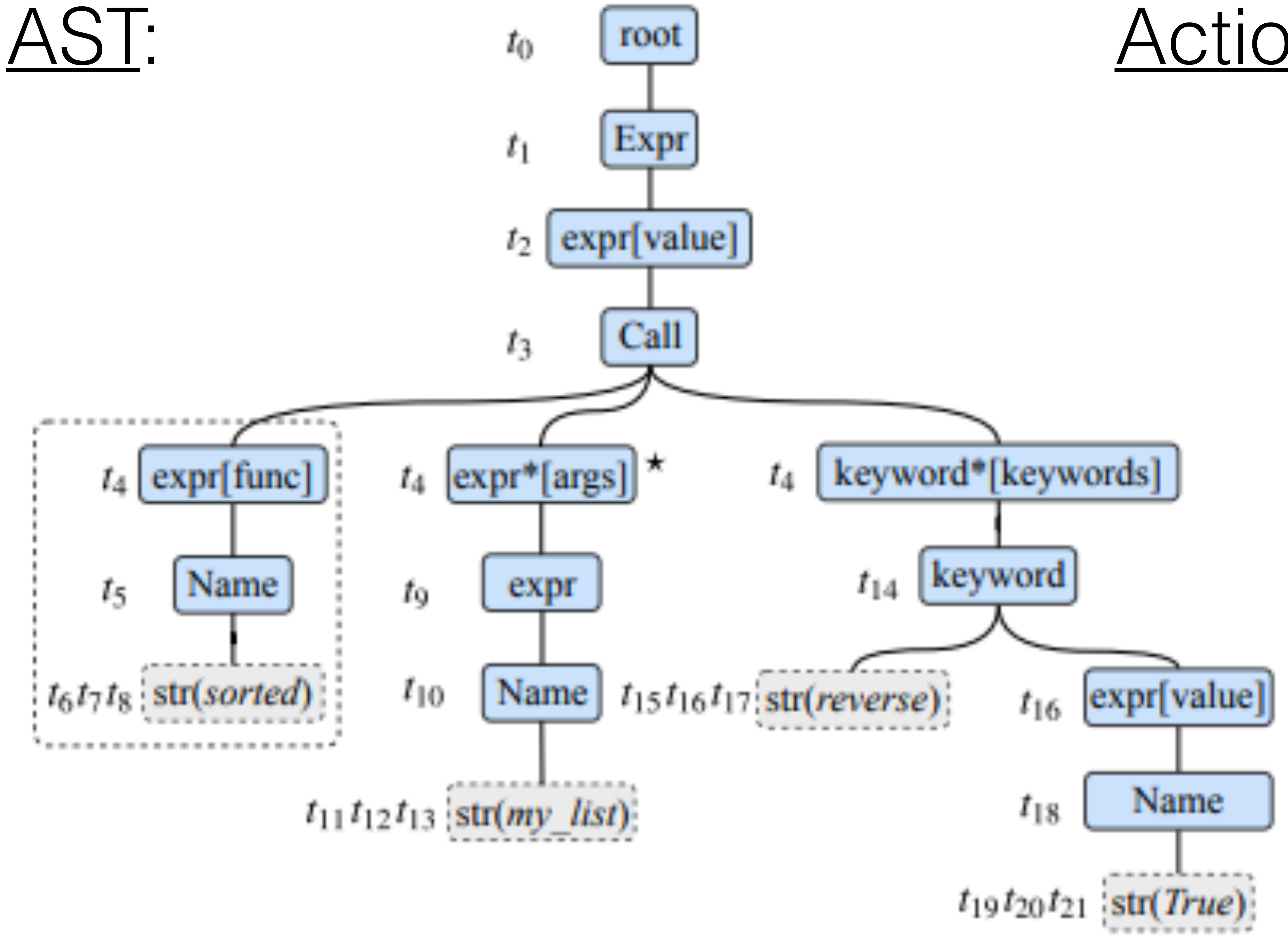
Image from Li Dong et al. Language to Logical Form with Neural Attention. ACL 2016

Generating Abstract Syntax Trees

Input: `sort my_list in a descending order`

Output code: `sorted(my_list, reverse=True)`

AST:



Action sequence:

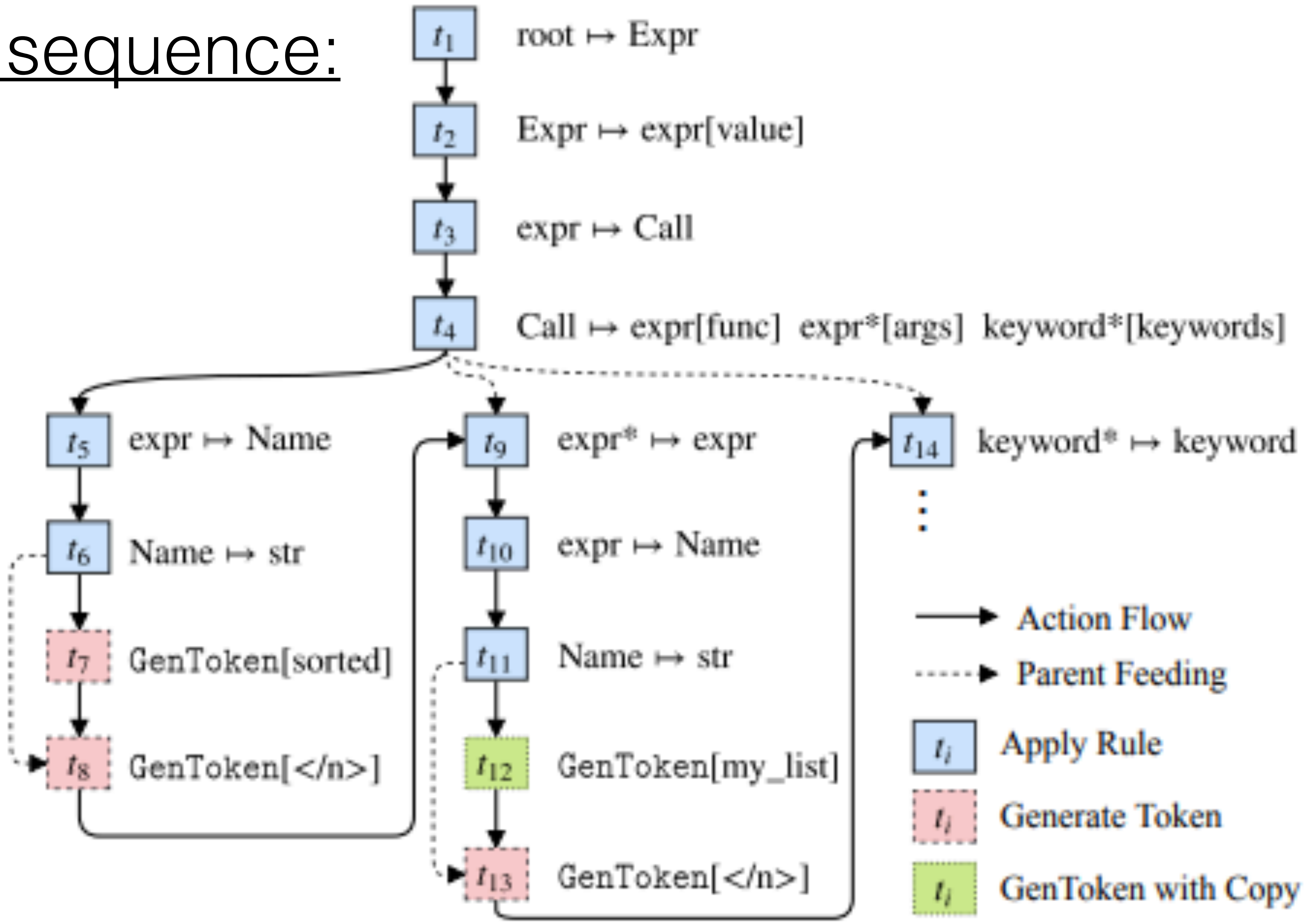
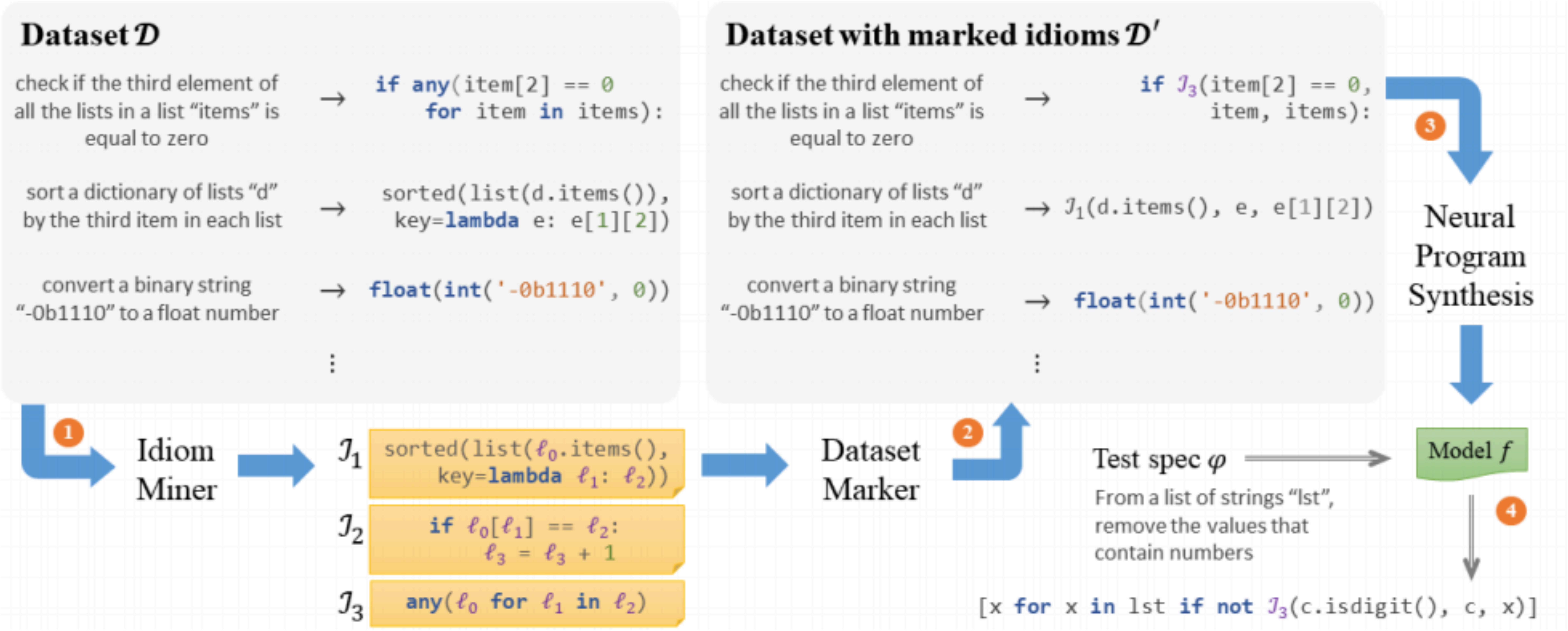


Image from Yin and Neubig. A Syntactic Neural Model for General-Purpose Code Generation. ACL 2017

Learning code idioms



Richard Shin et al. Program Synthesis and Semantic Parsing with Learned Code Idioms. NeurIPS 2019

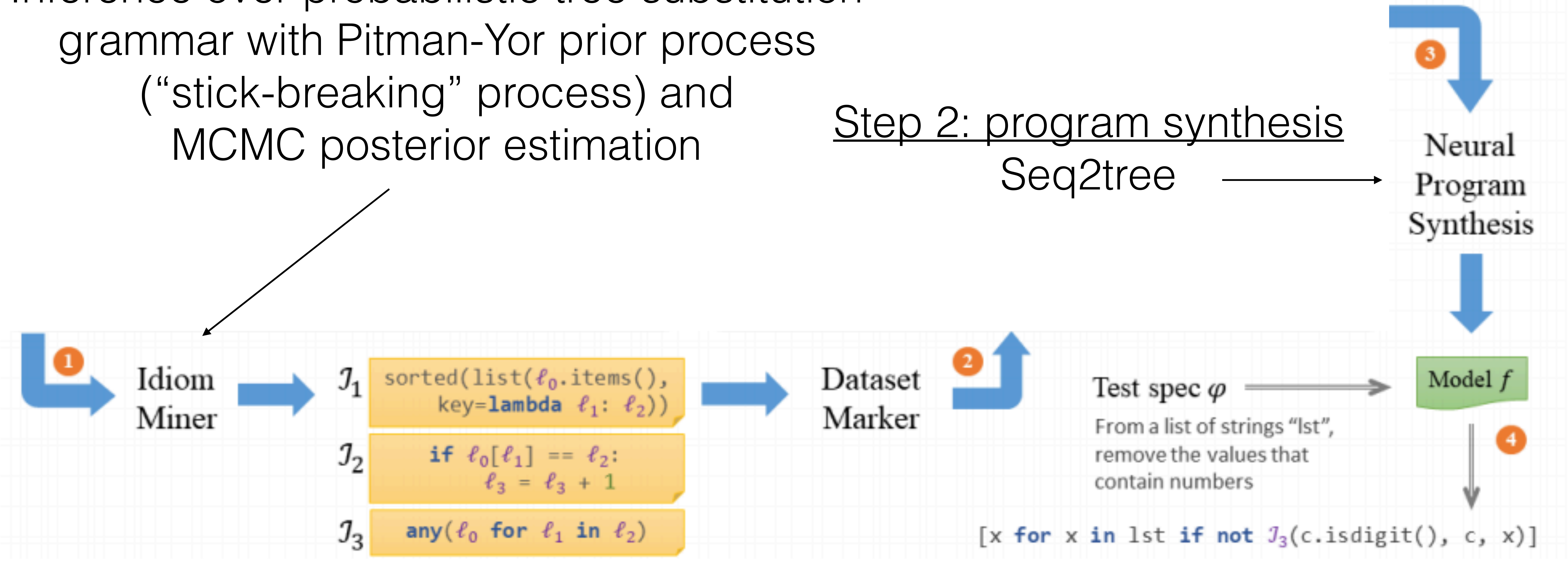
Learning code idioms

Step 1: idioms mining

Inference over probabilistic tree substitution grammar with Pitman-Yor prior process ("stick-breaking" process) and MCMC posterior estimation

Step 2: program synthesis

Seq2tree



Richard Shin et al. Program Synthesis and Semantic Parsing with Learned Code Idioms. NeurIPS 2019

Learning code idioms

Dataset: Spider (SQL)

K: number of idioms

Examples of mined idioms:

SELECT COUNT($\boxed{\ell_0 : \texttt{col}}$), $\boxed{\ell_1^*}$ WHERE $\boxed{\ell_2^*}$
INTERSECT $\boxed{\ell_4^? : \texttt{sql}}$ EXCEPT $\boxed{\ell_5^? : \texttt{sql}}$

WHERE $\boxed{\ell_0 : \texttt{col}}$ = $terminal

Quality

Model	K	Exact match
Baseline decoder	—	0.395
PATOIS, Score$_{\mathrm{Cov}}$	10	0.394
	20	0.379
	40	0.395
	80	0.407
PATOIS, Score$_{\mathrm{CXE}}$	10	0.368
	20	0.382
	40	0.387
	80	**0.416**

Richard Shin et al. Program Synthesis and Semantic Parsing with Learned Code Idioms. NeurIPS 2019

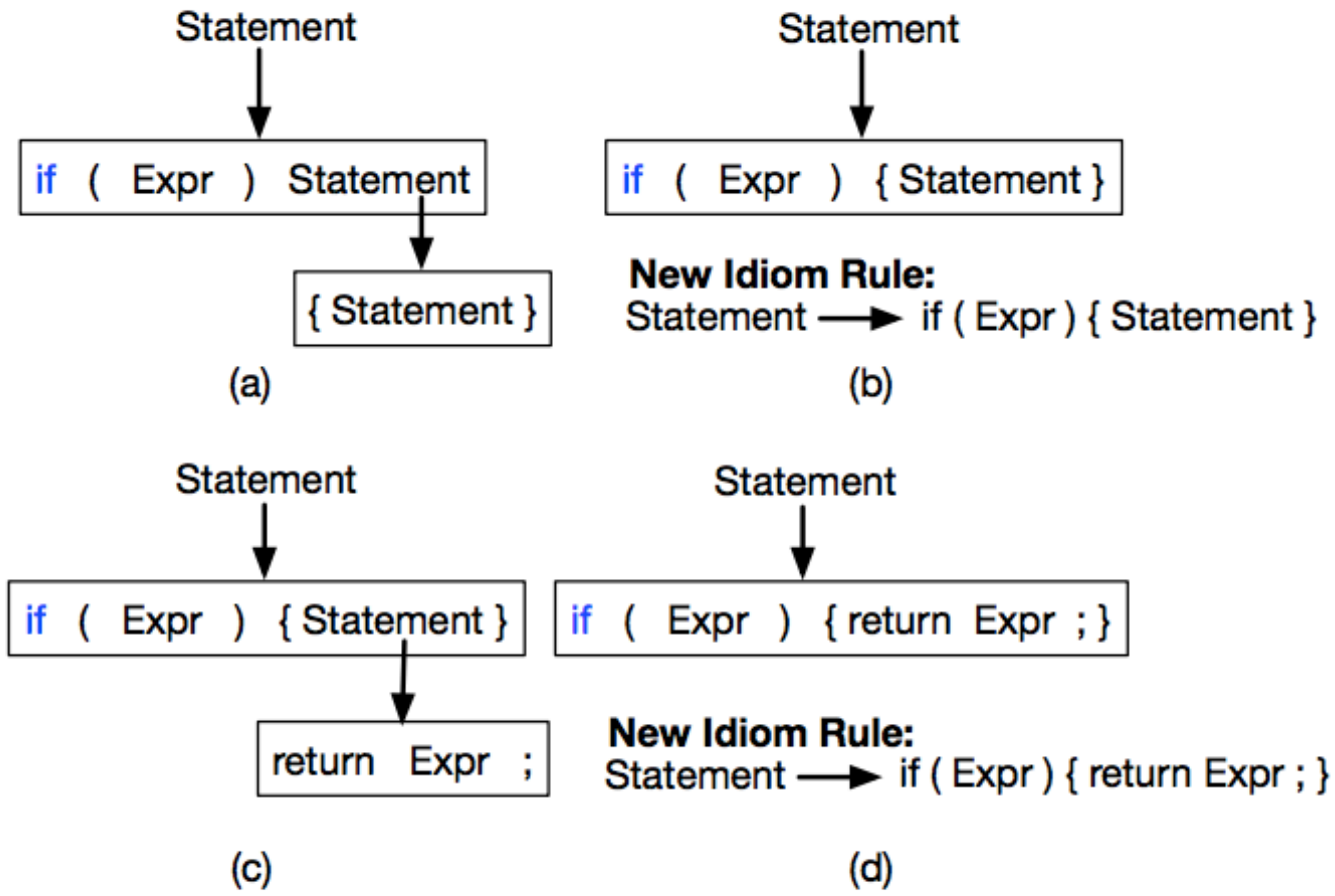
Learning code idioms - 2

Step 1: idioms mining

Byte-pair encoding (BPE) over trees:

at each step, find the most frequent depth-2 subtree,

until the desired number of idioms is reached



Step 2: program synthesis

Seq2tree

Srinivasan Iyer et al. Learning Programmatic Idioms for Scalable Semantic Parsing. EMNLP 2019

Learning code idioms - 2

Dataset: CONCODE
(Java with context)

Effect of using idioms:

Model	Exact	BLEU	Training Time (h)
Iyer-Simp	9.8	23.2	27
+ 100 idioms	9.8	24.5	15
+ 200 idioms	9.8	24.0	13
+ 300 idioms	9.6	23.8	12
+ 400 idioms	9.7	23.8	11
+ 600 idioms	9.9	22.7	11

Srinivasan Iyer et al. Learning Programmatic Idioms for Scalable Semantic Parsing. EMNLP 2019

Learning code idioms - 2

Dataset: CONCODE
(Java with context)

Effect of using idioms:

Model	Exact	BLEU	Training Time (h)
Iyer-Simp	9.8	23.2	27
+ 100 idioms	9.8	24.5	15
+ 200 idioms	9.8	24.0	13
+ 300 idioms	9.6	23.8	12
+ 400 idioms	9.7	23.8	11
+ 600 idioms	9.9	22.7	11

Examples of idioms:

```
Expr → new Identifier
         ( ExprList )

Statement →
  try Block
  catch ( Identifier Identifier )
  Block

Statement →
  if ( ParExpr )
  { throw new
      Identifier (“String“) ; }
```

Srinivasan Iyer et al. Learning Programmatic Idioms for Scalable Semantic Parsing. EMNLP 2019

Learning code idioms - 2

Dataset: CONCODE
(Java with context)

Effect of using idioms:

Model	Exact	BLEU	Training Time (h)
Iyer-Simp	9.8	23.2	27
+ 100 idioms	9.8	24.5	15
+ 200 idioms	9.8	24.0	13
+ 300 idioms	9.6	23.8	12
+ 400 idioms	9.7	23.8	11
+ 600 idioms	9.9	22.7	11

Examples of idioms:

Expr → new Identifier (ExprList)

Statement → try Block catch (Identifier Identifier) Block

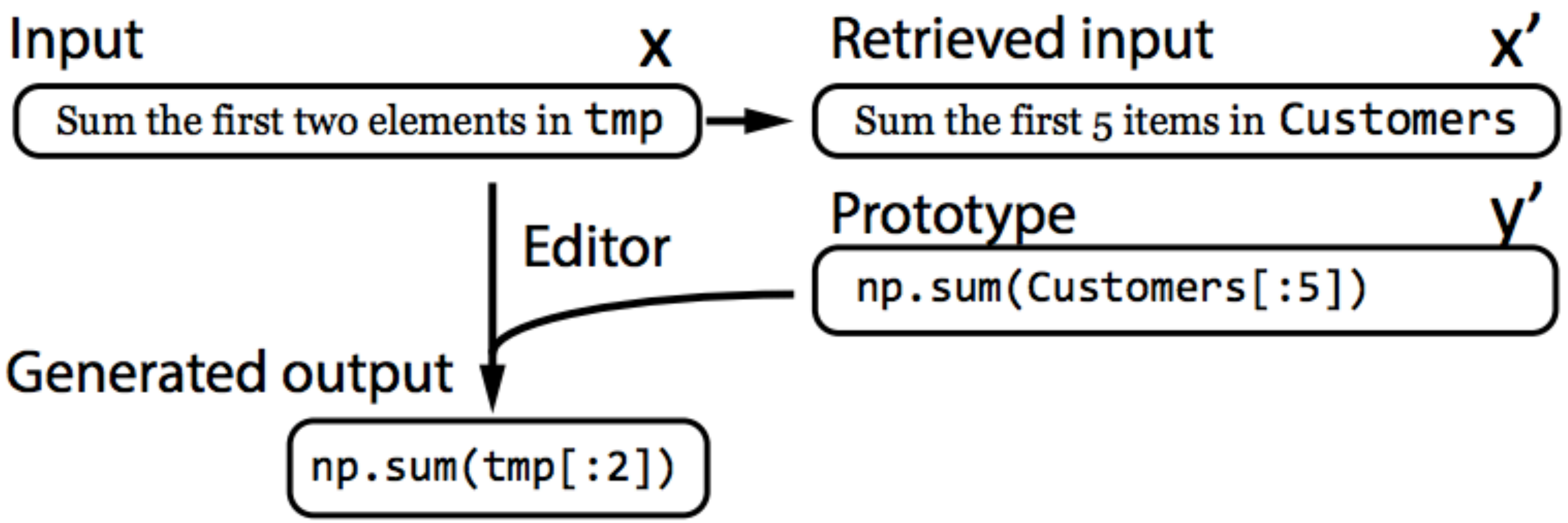
Statement → if (ParExpr) { throw new Identifier ("String") ; }

Effect of including longer inputs in training:

Model	Exact	BLEU
1× Train	12.0 (9.7)	26.3 (23.8)
2× Train	13.0 (10.3)	28.4 (25.2)
3× Train	13.3 (10.4)	28.6 (26.5)
5× Train	13.4 (11.0)	28.9 (26.6)

Srinivasan Iyer et al. Learning Programmatic Idioms for Scalable Semantic Parsing. EMNLP 2019

Retrieve&edit approach



	BLEU
Retrieve-and-edit (Retrieve+Edit)	**34.7**
Seq2Seq	19.2
Retriever only (TaskRetriever)	29.9

1. Train standard Seq2seq model:
   text → *embedding* → code
2. Retriever = 1NN over *embeddings*
3. Train Seq2Seq editor:
   prototype → code

Dataset: Python autocompletion

+ follow-up Guo et al, ACL'19 with experiments on CONCODE dataset

Tatsunori B. Hashimoto et al. A Retrieve-and-Edit Framework for Predicting Structured Outputs. NeurIPS 2018

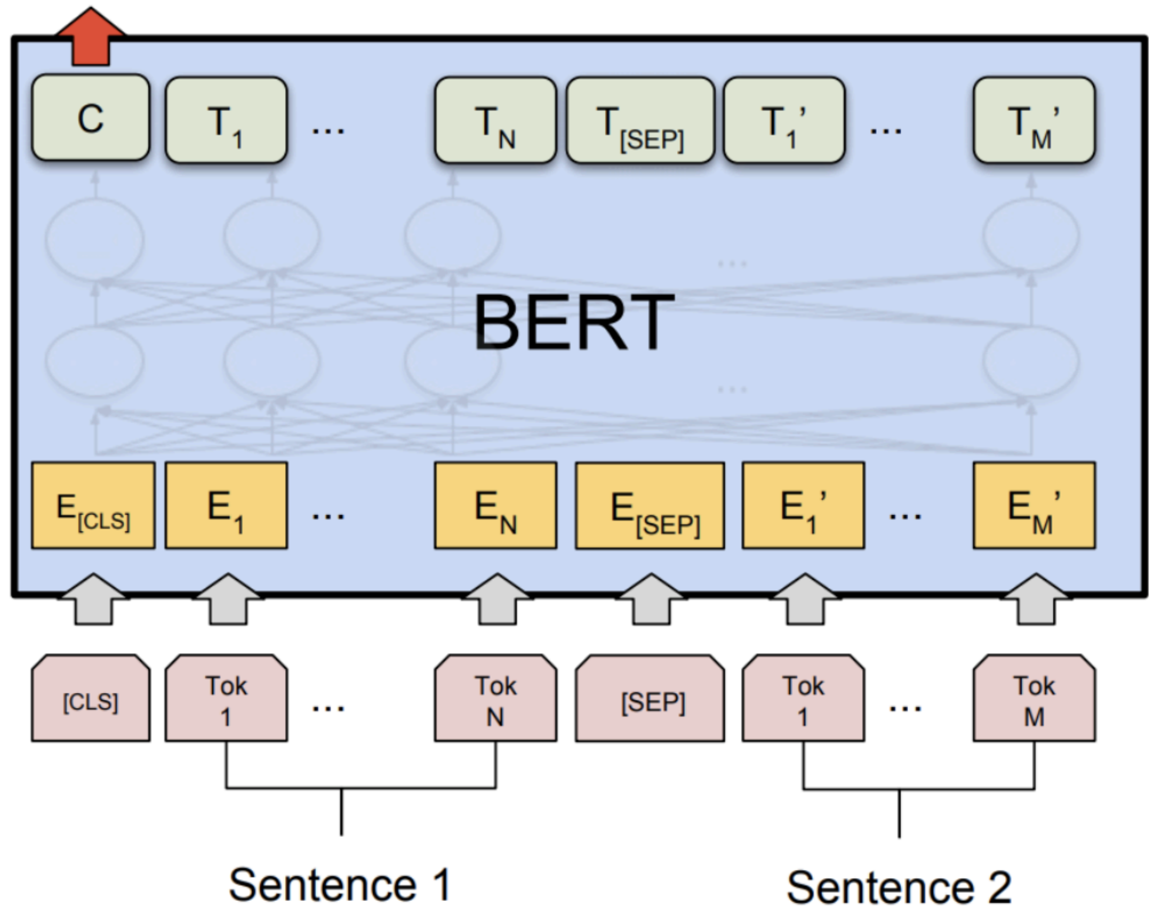
Outline

- Problem statement, datasets, metrics
- Approaches
  - Supervised learning-based
  - Self-supervised pretraining-based
- Remaining challenges
- Codex: details, demos, experiments - next week

Pretraining-based approaches

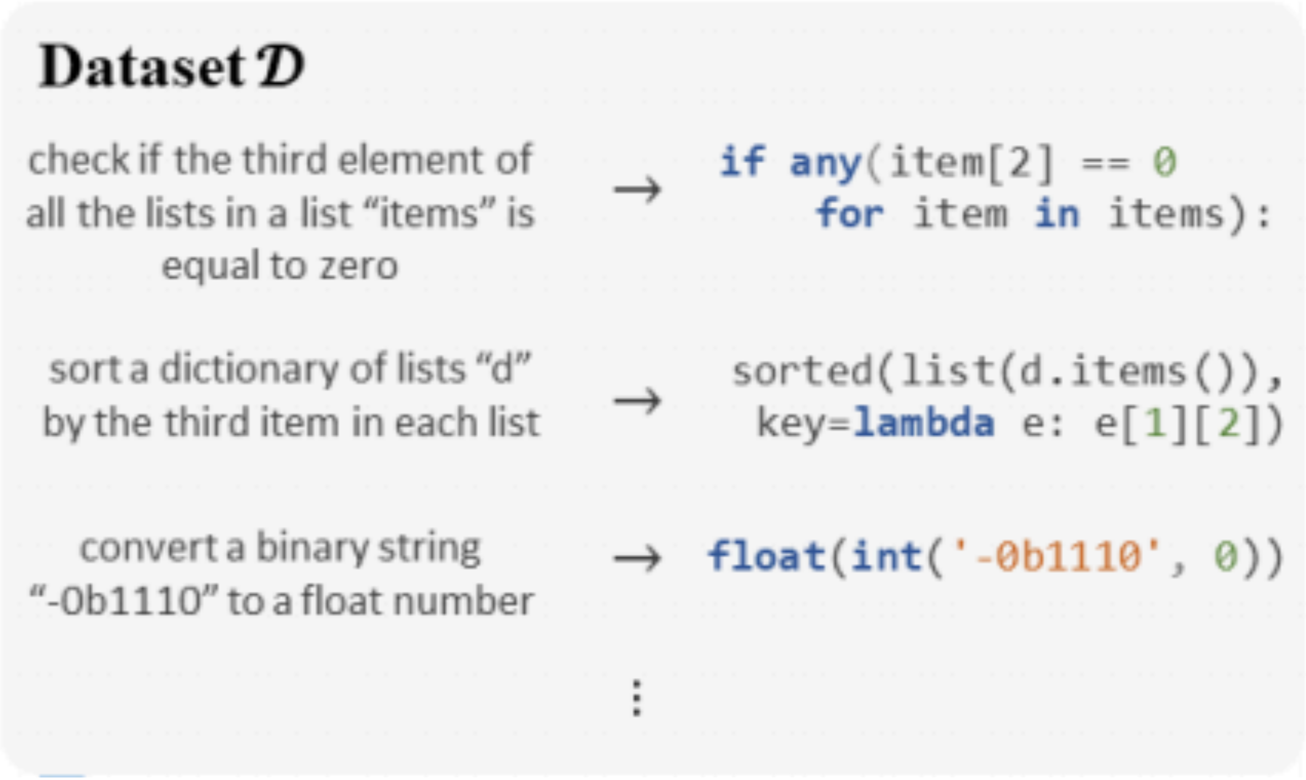
GitHub

self-supervised pretraining



Final model

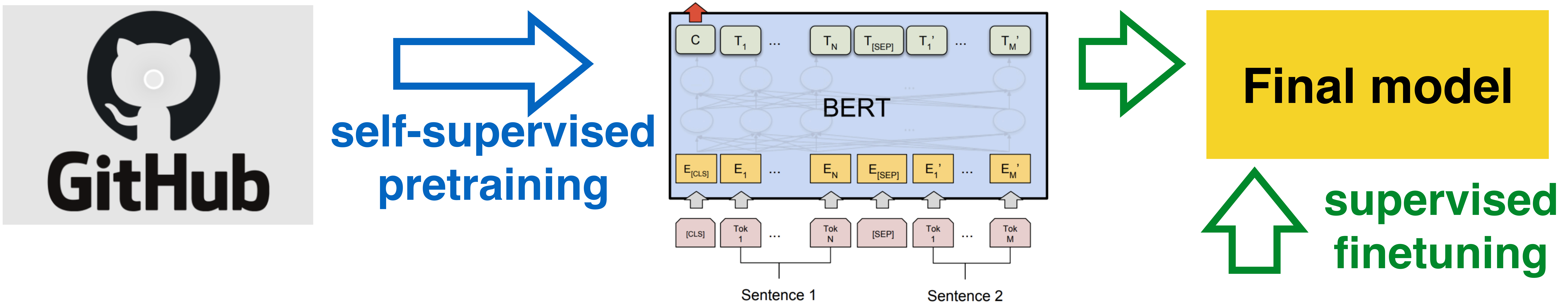
supervised finetuning



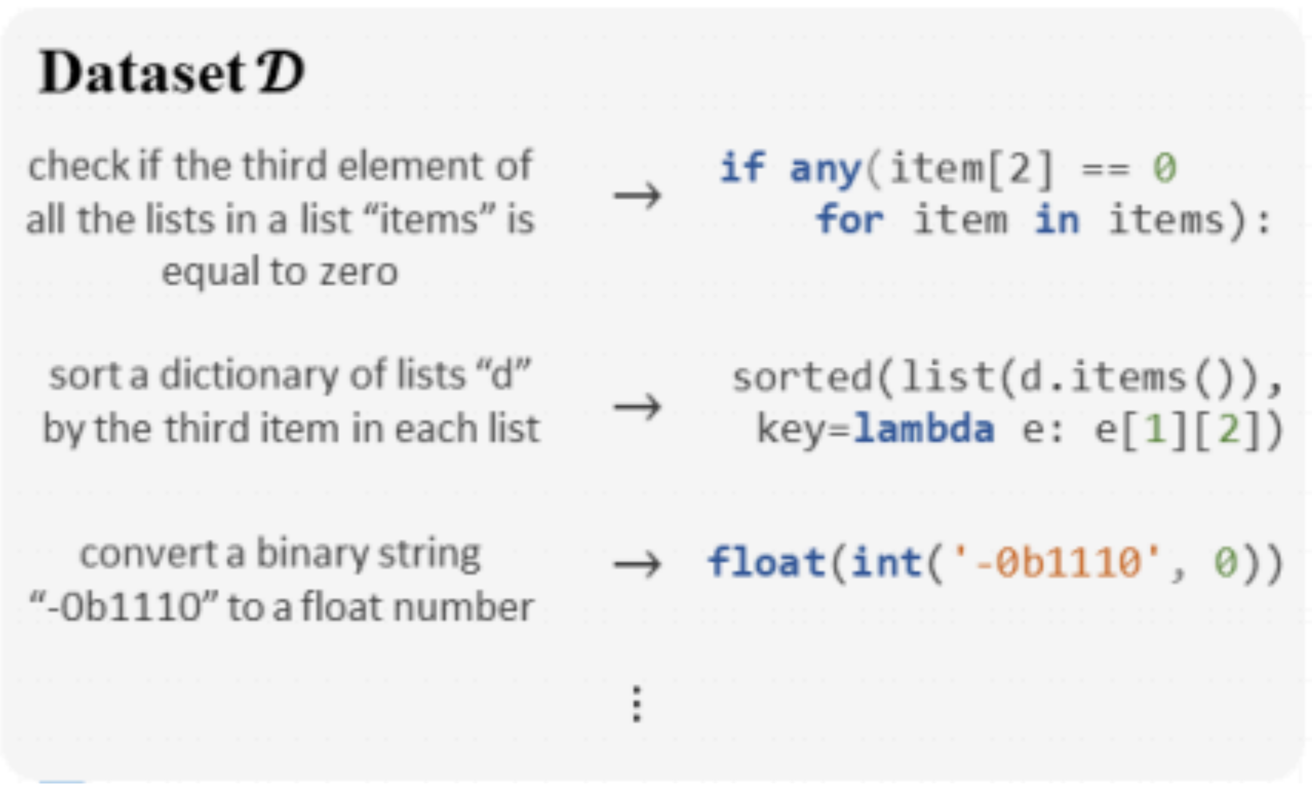
Dataset $\mathcal{D}$

check if the third element of all the lists in a list "items" is equal to zero	→	`if any(item[2] == 0 for item in items):`
sort a dictionary of lists "d" by the third item in each list	→	`sorted(list(d.items()), key=lambda e: e[1][2])`
convert a binary string "-0b1110" to a float number	→	`float(int('-0b1110', 0))`
	⋮	

Pretraining-based approaches



Model	Pretraining task	Supported languages	Architecture
CodeBERT (Feng et al, EMNLP'20)	BERT+RTD	6 langs	Encoder-only
CuBERT (Kanade et al., ICML'20)	BERT	Python	Encoder-only
CodeGPT-2 (Lu et al, arxiv'21)	GPT	6 langs	Decoder-only
PLBART (Ahmad et al, NAACL'20)	BART	6 langs	Encoder-decoder



No need for parsing ASTs, just treating code as a sequence of tokens

Program synthesis results from PLBART
(encoder-decoder)

Illustration of pretraining tasks

	PLBART Encoder Input	PLBART Decoder Output
Token masking:	Is 0 the **[MASK]** Fibonacci **[MASK]** ? <En>	<En> Is 0 the **first** Fibonacci **number** ?
Token deletion:	public static main (String args [ ]) { date = Date ( ) ; System . out . (String . format (" Current Date : % tc " ,)) ; } <java>	<java> public static **void** main (String args [ ]) { **Date** date = **new** Date ( ) ; System . out . **printf** (String . format (" Current Date : % tc " , **date**)) ; }
Token infilling:	def addThreeNumbers (x , y , z) : NEW_LINE INDENT return **[MASK]** <python>	<python> def addThreeNumbers (x , y , z) : NEW_LINE INDENT return **x + y + z**

Ahmad et al. Unified Pre-training for Program Understanding and Generation. NAACL 2021

Program synthesis results from PLBART
(encoder-decoder)

Illustration of pretraining tasks

	PLBART Encoder Input	PLBART Decoder Output
Token masking:	Is 0 the **[MASK]** Fibonacci **[MASK]** ? <En>	<En> Is 0 the **first** Fibonacci **number** ?
Token deletion:	public static main (String args [ ]) { date = Date ( ) ; System . out . (String . format (" Current Date : % tc " ,)) ; } <java>	<java> public static **void** main (String args [ ]) { **Date** date = **new** Date ( ) ; System . out . **printf** (String . format (" Current Date : % tc " , **date**)) ; }
Token infilling:	def addThreeNumbers (x , y , z) : NEW_LINE INDENT return **[MASK]** <python>	<python> def addThreeNumbers (x , y , z) : NEW_LINE INDENT return **x + y + z**

Dataset: CONCODE
(Java with context)

Methods	EM	BLEU	CodeBLEU
Seq2Seq	3.05	21.31	26.39
Guo et al. (2019)	10.05	24.40	29.46
Iyer et al. (2019)	12.20	26.60	-
GPT-2	17.35	25.37	29.69
CodeGPT-2	18.25	28.69	32.71
CodeGPT-adapted	**20.10**	32.79	35.98
PLBART	18.75	**36.69**	**38.52**

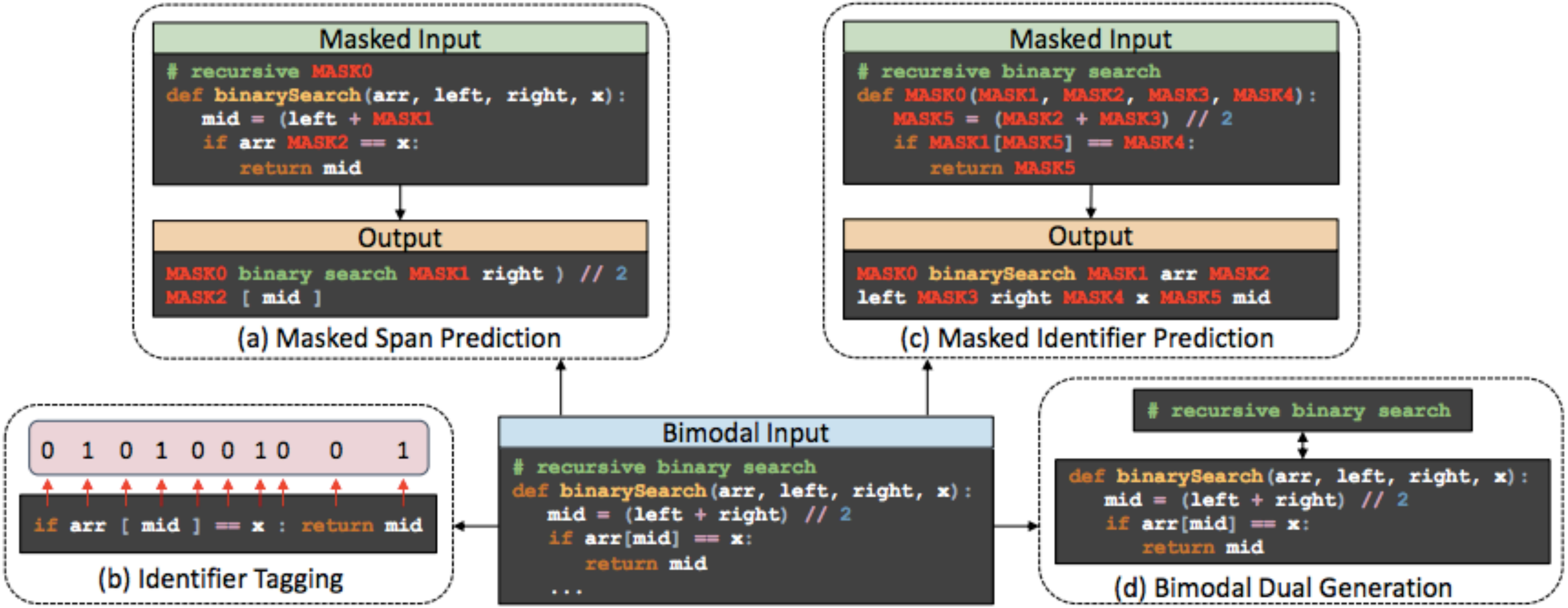
No domain-specific inductive bias, pure NLP techniques!

Ahmad et al. Unified Pre-training for Program Understanding and Generation. NAACL 2021

CodeT5

(encoder-decoder)

Illustration of pretraining tasks



Yue Wang et al. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. EMNLP'21

CodeT5
(encoder-decoder)

Dataset: CONCODE
(Java with context)

Methods	EM	BLEU	CodeBLEU
GPT-2	17.35	25.37	29.69
CodeGPT-2	18.25	28.69	32.71
CodeGPT-adapted	20.10	32.79	35.98
PLBART	18.75	36.69	38.52
CodeT5-base	22.30	40.73	43.20
+dual-gen	**22.70**	**41.48**	**44.10**

Yue Wang et al. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. EMNLP'21

CodeT5
(encoder-decoder)

Dataset: CONCODE
(Java with context)

Methods	EM	BLEU	CodeBLEU
GPT-2	17.35	25.37	29.69
CodeGPT-2	18.25	28.69	32.71
CodeGPT-adapted	20.10	32.79	35.98
PLBART	18.75	36.69	38.52
CodeT5-base	22.30	40.73	43.20
+dual-gen	**22.70**	**41.48**	**44.10**

Improved prediction
of identifiers:

Type	Code
Source	**Text:** returns the string value of the specified field. the value is obtained from whichever scan contains the field. **Env:** Scan s1 ; Scan s2 ; boolean hasField
CodeT5	`String function (String arg0){ if (s1.hasField(arg0)) return s1.getString(arg0); else return s2.getString(arg0);}`
W/o MIP+IT	`String function (String arg0){ return s1.getString(arg0);}`

Yue Wang et al. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. EMNLP'21

CodeT5
(encoder-decoder)

Dataset: CONCODE
(Java with context)

Methods	EM	BLEU	CodeBLEU
GPT-2	17.35	25.37	29.69
CodeGPT-2	18.25	28.69	32.71
CodeGPT-adapted	20.10	32.79	35.98
PLBART	18.75	36.69	38.52
CodeT5-base	22.30	40.73	43.20
+dual-gen	**22.70**	**41.48**	**44.10**

Improved prediction
of identifiers:

Type	Code
Source	**Text:** `returns the string value of the specified field. the value is obtained from whichever scan contains the field.` **Env:** `Scan s1; Scan s2; boolean hasField`
CodeT5	`String function (String arg0){ if (s1.hasField(arg0)) return s1.getString(arg0); else return s2.getString(arg0);}`
W/o MIP+IT	`String function (String arg0){ return s1.getString(arg0);}`

Ablation
study:

Methods	Code-Gen (CodeBLEU)
CodeT5	41.39
-MSP	37.44
-IT	39.21
-MIP	38.25

Yue Wang et al. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. EMNLP'21

ContraCode pretraining
(encoder-only)

```
function x(maxLine) {
  const section = {
    text: '',
    data
  };

  for (; i < maxLine; i += 1) {
    section.text += `${lines[i]}\n`;
  }

  if (section) {
    parsingCtx.sections.push(section);
  }
}
```

Original JavaScript method

→

```
function x(t) {
  const n = {
    'text': '',
    'data': data
  };
  for (;i < t; i += 1) {
    n.text += lines[i] + '\n';
  }
  n && parsingCtx.sections.push(n);
}
```

Renamed variables, explicit object style, explicit concatenation, inline conditional

Paras Jain et al. Contrastive Code Representation Learning. arxiv 2020

ContraCode pretraining
(encoder-only)

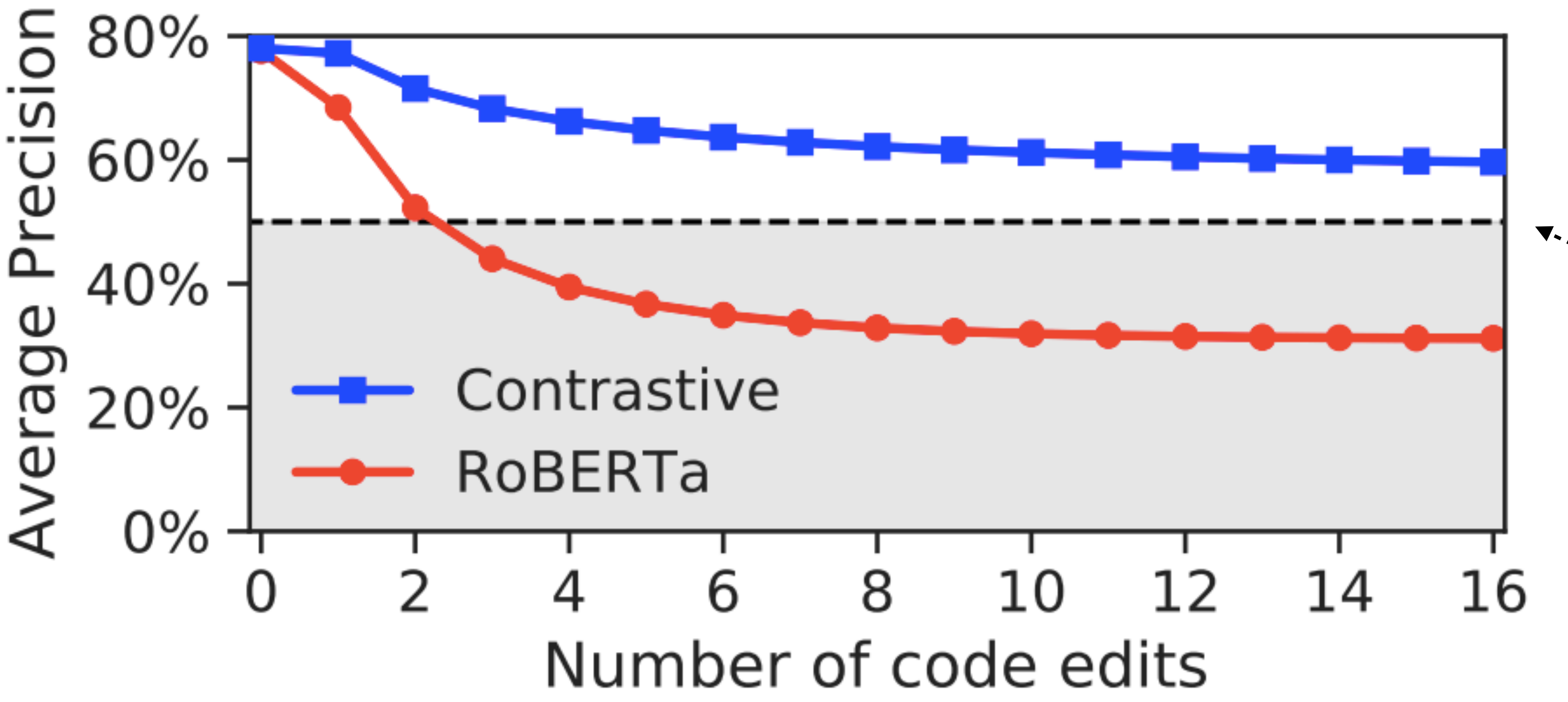
Task: type prediction (no experiments on program synthesis in the paper)

Baseline	Method	Acc@1 (all types)	Acc@5 (all types)
Static analysis	TypeScript CheckJS (Bierman et al., 2014)	45.11%	45.11%
	Name only (Hellendoorn et al., 2018)	28.94%	70.07%
Transformer	Transformer (supervised)	45.66%	80.08%
	with ContraCode pre-training	**46.86%**	**81.85%**
RoBERTa	Transformer (RoBERTa MLM pre-training)	40.85%	75.76%
	with ContraCode pre-training	**47.16%**	**81.44%**
DeepTyper (BiLSTM)	DeepTyper (supervised)	51.73%	82.71%
	with RoBERTa MLM pre-training (10K steps)	50.24%	82.85%
	with ContraCode pre-training	52.65%	84.60%
	with ContraCode pre-training (w/ subword reg. ft.)	**54.01%**	**85.55%**

Paras Jain et al. Contrastive Code Representation Learning 2020

ContraCode pretraining
(encoder-only)

Robustness to simple label-preserving code edits, e. g. variable renaming:



random guess performance

Task: clone detection

Paras Jain et al. Contrastive Code Representation Learning 2020

Codex
(decoder-only)

```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)
```

Mark Chen et al. Evaluating Large Language Models Trained on Code. arxiv 2021

Outline

- Problem statement, datasets, metrics
- Approaches
  - Supervised learning-based
  - Self-supervised pretraining-based
- Remaining challenges
- Codex: details, demos, experiments - next week

Performance of pretrained models on APPS dataset

mean number of passed tests

portion of problems with **all** tests passed

Model	Test Case Average Introductory	Interview	Competitive	Average	Strict Accuracy Introductory	Interview	Competition	Average
GPT-2 0.1B	5.64	6.93	4.37	6.16	1.00	0.33	0.00	0.40
GPT-2 1.5B	7.40	9.11	5.05	7.96	1.30	0.70	0.00	0.68
GPT-Neo 2.7B	14.68	9.85	6.54	10.15	3.90	0.57	0.00	1.12
GPT-3 175B	0.57	0.65	0.21	0.55	0.20	0.03	0.00	0.06

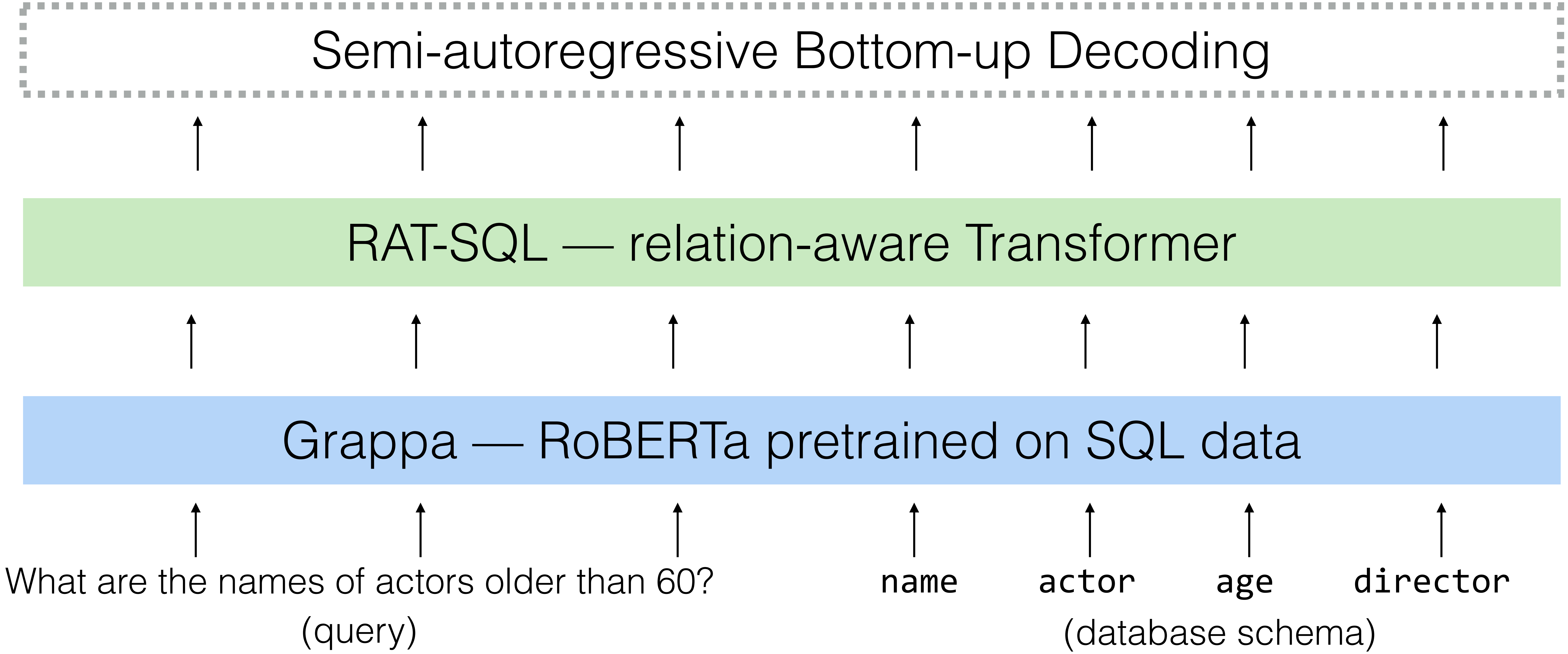
model pretrained on texts (access through API)

initialization with text-based model
and pretraining on GitHub

Dan Hendrycks et al. Measuring Coding Challenge Competence With APPS. arXiv 2021

Recent state-of-the-art model on Spider dataset

SMBOP: Semi-autoregressive Bottom-up Semantic Parsing (SQL dataset)

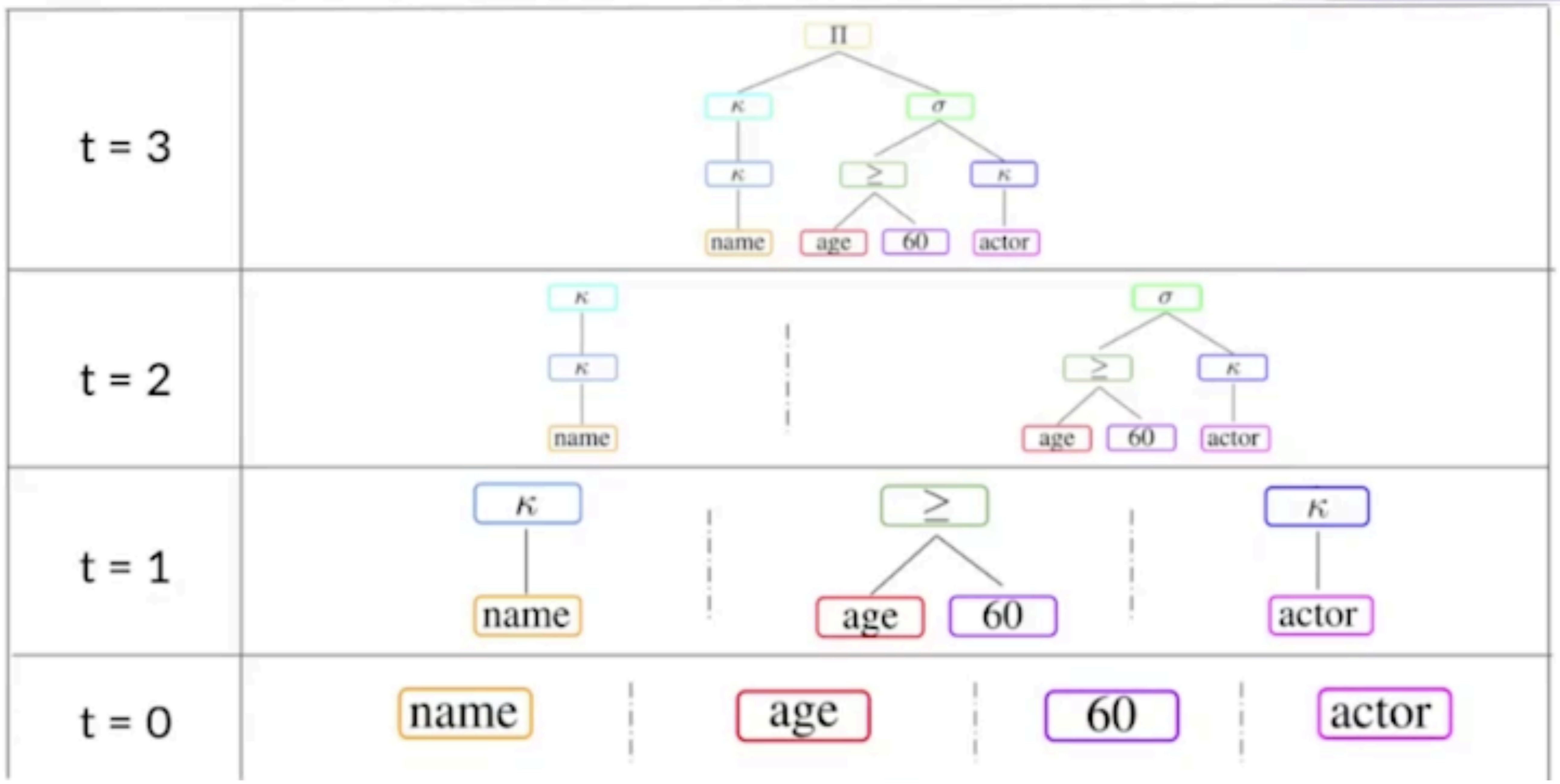


Ohad Rubin and Jonathan Berant. SMBOP: Semi-autoregressive Bottom-up Semantic Parsing. NAACL 2021

Recent state-of-the-art model on Spider dataset

SMBOP: Semi-autoregressive Bottom-up Semantic Parsing (SQL dataset)

Semi-autoregressive Bottom-up Decoding



Not everything is done with pretraining!

Ohad Rubin and Jonathan Berant. SMBOP: Semi-autoregressive Bottom-up Semantic Parsing. NAACL 2021

Summary

- From domain-specific architectures to pretraining-based approaches

- Challenging datasets: APPS, Spider

- Moving towards running tests on generated code rather than using BLEU

Next week: Codex: details, demos and experiments