

Neural & Neuro-Symbolic Program Synthesis

Alex Polozov

X, the moonshot factory (formerly Google X)

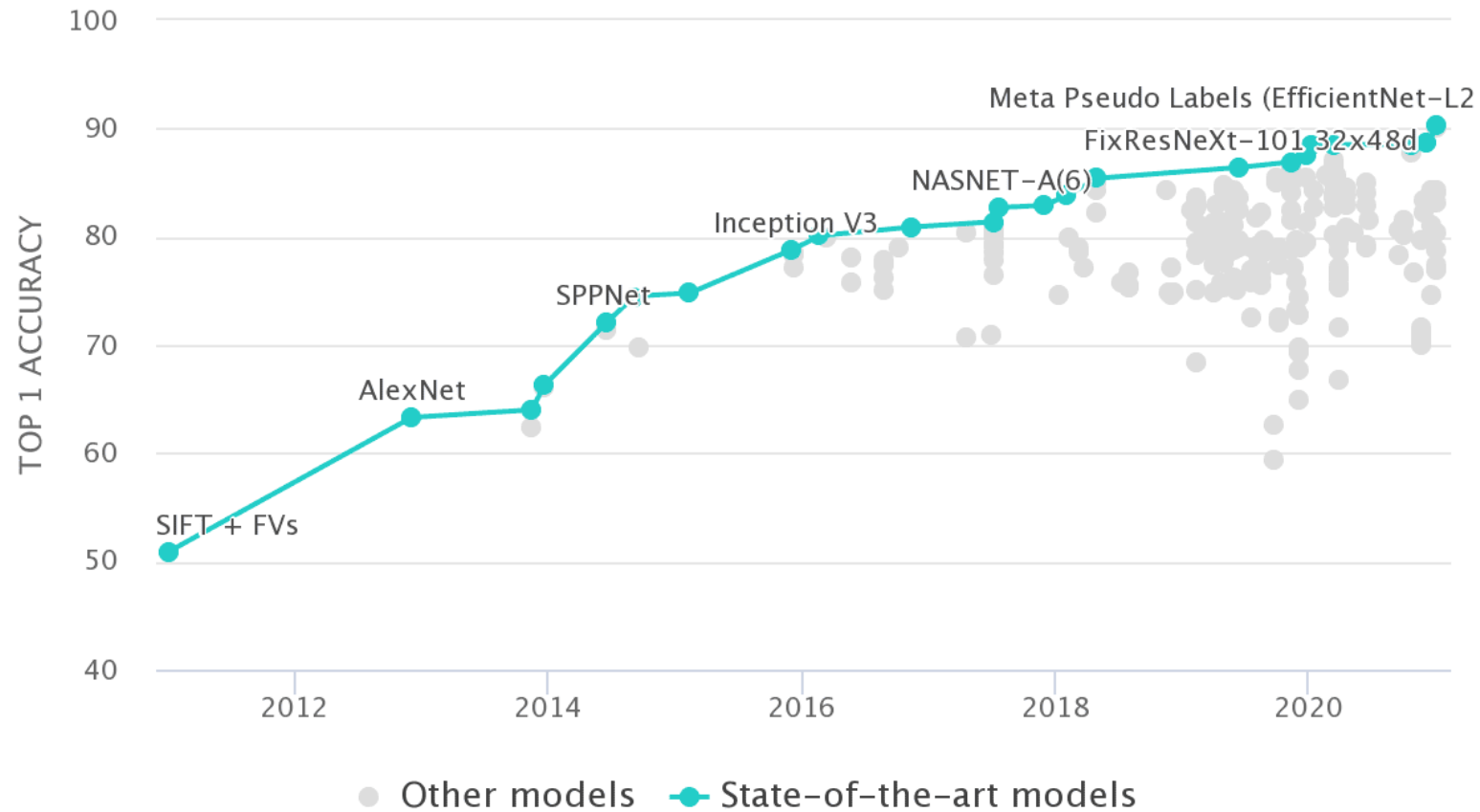
polozov@x.team

Outline

- Why deep learning?
- Neural-guided program synthesis
- Neural code generation
 - Informal specs
 - Execution guidance
- Modern neural methods
 - Transformer language models
 - Neuro-symbolic program synthesis

Logistics: lecture today, rest of lecture + paper discussion Thursday.

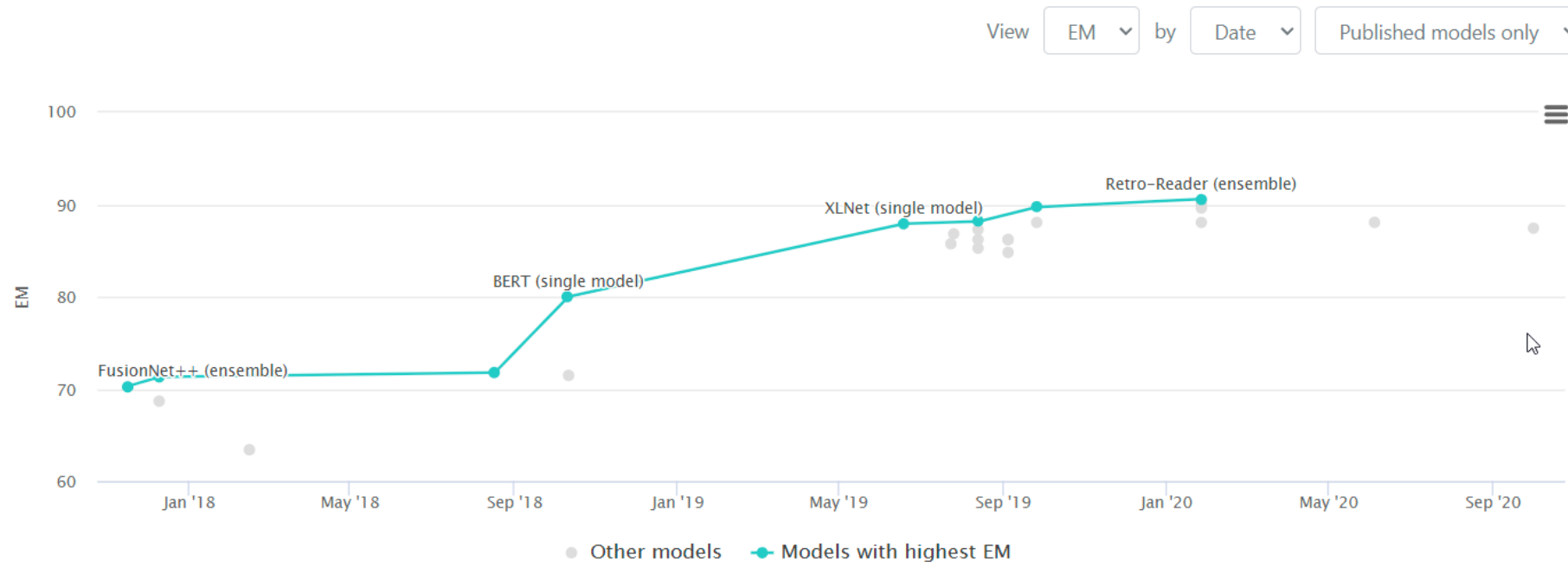
The Deep Learning Revolution: Vision



The Deep Learning Revolution: NLP

Question Answering on SQuAD2.0

Leaderboard



The Deep Learning Revolution

What changed?

- Large-scale dataset availability (e.g., ImageNet, web texts)
- Parallel compute (GPUs, TPUs) and learning methods that use it
 - Enumerative search or older ML methods are CPU-bound! Harder to parallelize.
- Frameworks for describing learning problems in regular code (“differentiable programming”)



Andrej Karpathy

Nov 11, 2017 · 9 min read

Software 2.0

I sometimes see people refer to neural networks as just “another tool in your machine learning toolbox”. They have some pros and cons, they work here or there, and sometimes you can use them to win Kaggle competitions.

Unfortunately, this interpretation completely misses the forest for the trees.

Neural networks are not just another classifier, they represent the beginning of a fundamental shift in how we develop software. They are Software 2.0.

Usage in program synthesis

 Search guidance

 Likelihood models

 Informal specifications

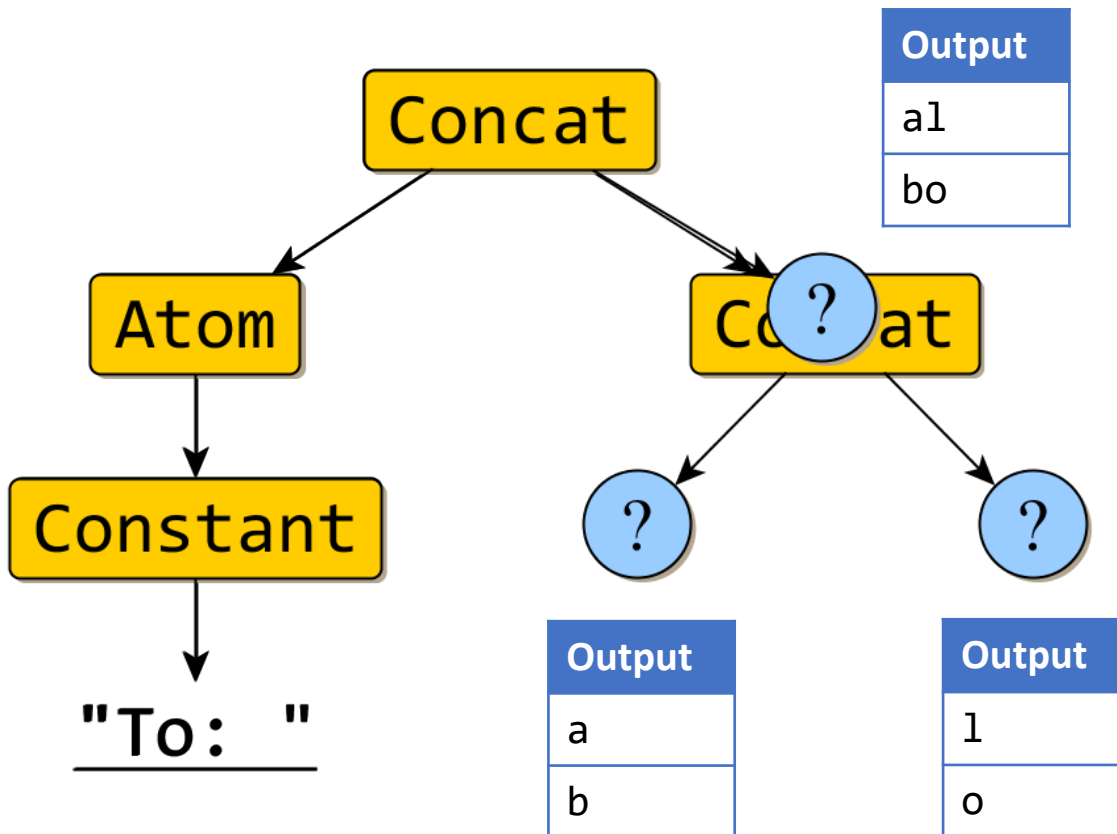


Search Guidance

Neural search guidance

- Recall: weighted top-down search (DeepCoder and Euphony)
- Papers:
 - Balog, M., A. L. Gaunt, M. Brockschmidt, S. Nowozin, and D. Tarlow. "DeepCoder: Learning to Write Programs." In *International Conference on Learning Representations (ICLR 2017)*.
 - Kalyan, A., Mohta, A., Polozov, O., Batra, D., Jain, P., & Gulwani, S. Neural-Guided Deductive Search for Real-Time Program Synthesis from Examples. In *International Conference on Learning Representations (ICLR 2018)*.

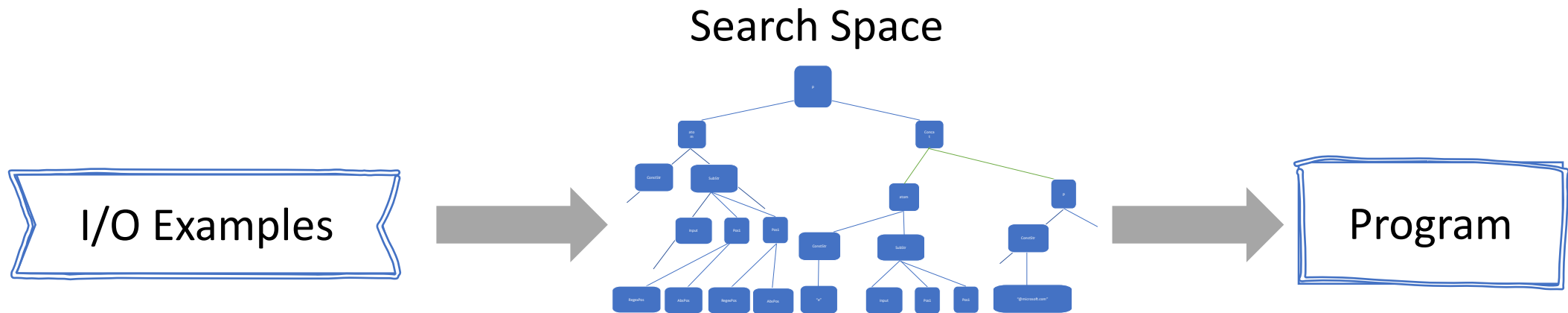
Deductive Search



Input	Output
alice liddell	To: al
bob o'reilly	To: bo

1. Select a hole.
 2. Select an operator to expand.
 3. Propagate the examples.
- ✓ Correct by construction
 - ✓ Constraint propagation exists
for many operations & domains
 - ✓ Easy to add a ranking function
 - ✗ Exponentially slow

Deductive Search



Why so slow? Explores the entire search space
(unless deduction prunes some of it)

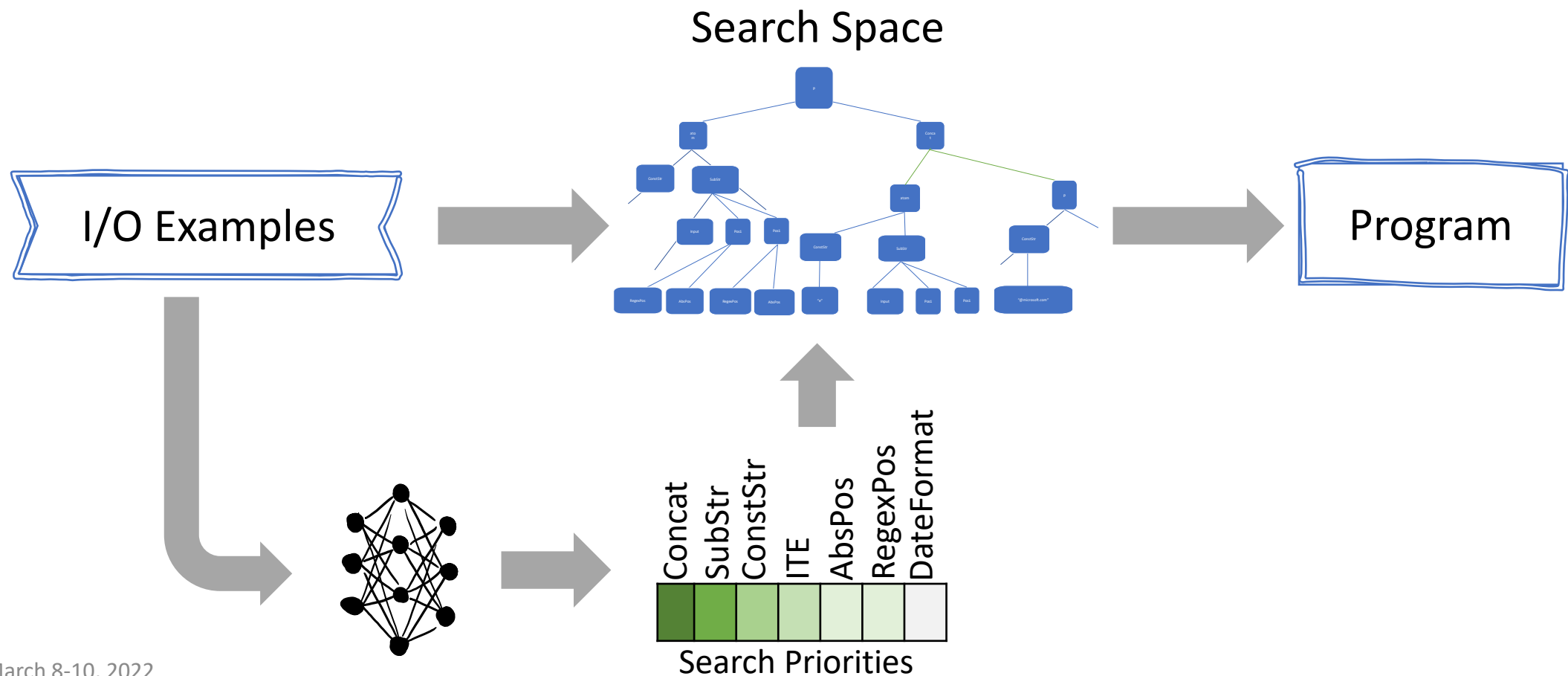
Machine-learned insights

Input	Output
alice liddell	To: al
bob o'reilly	To: bo

Can't be a substring, requires concatenation

[Balog et al., 2017]

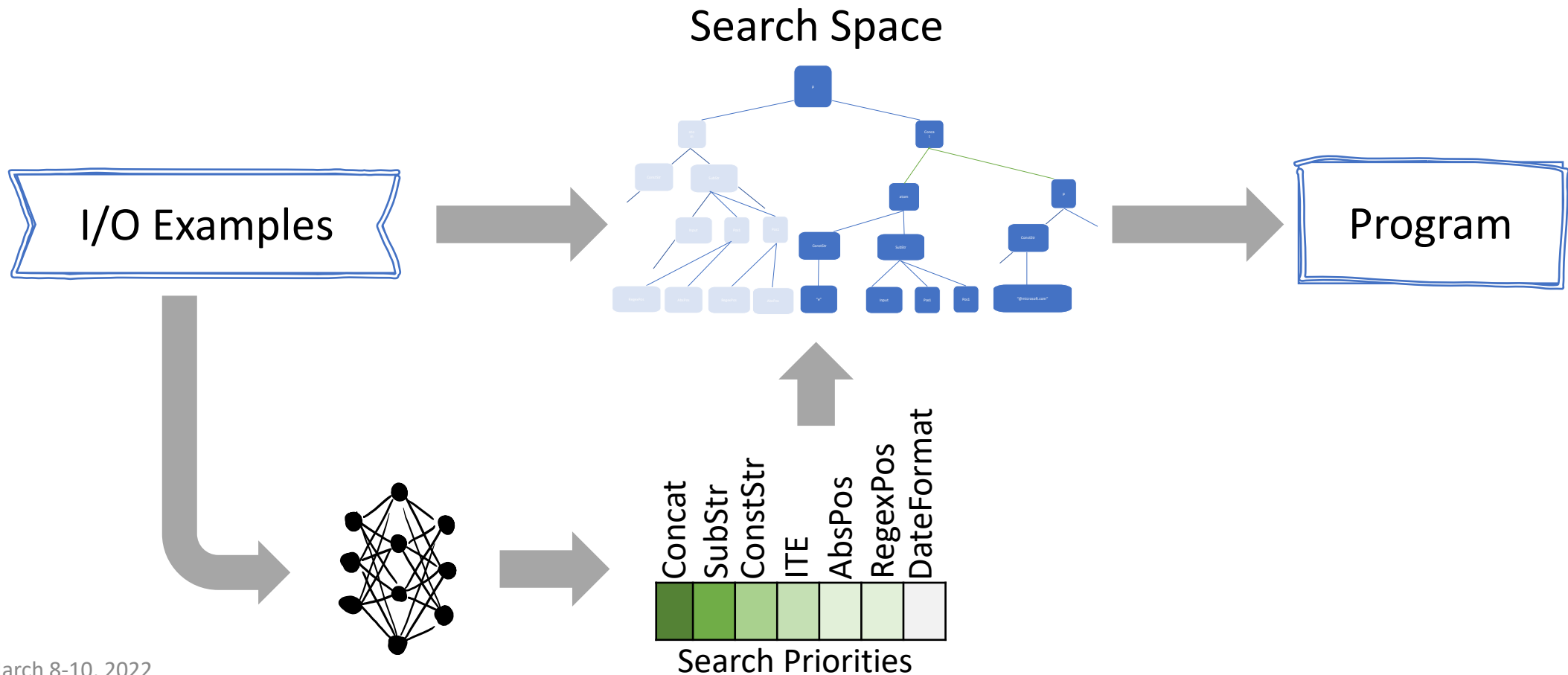
Idea: Order the search space based on a priority list from DNN *before starting*



DeepCoder: Learning to Write Programs

[Balog et al., 2017]

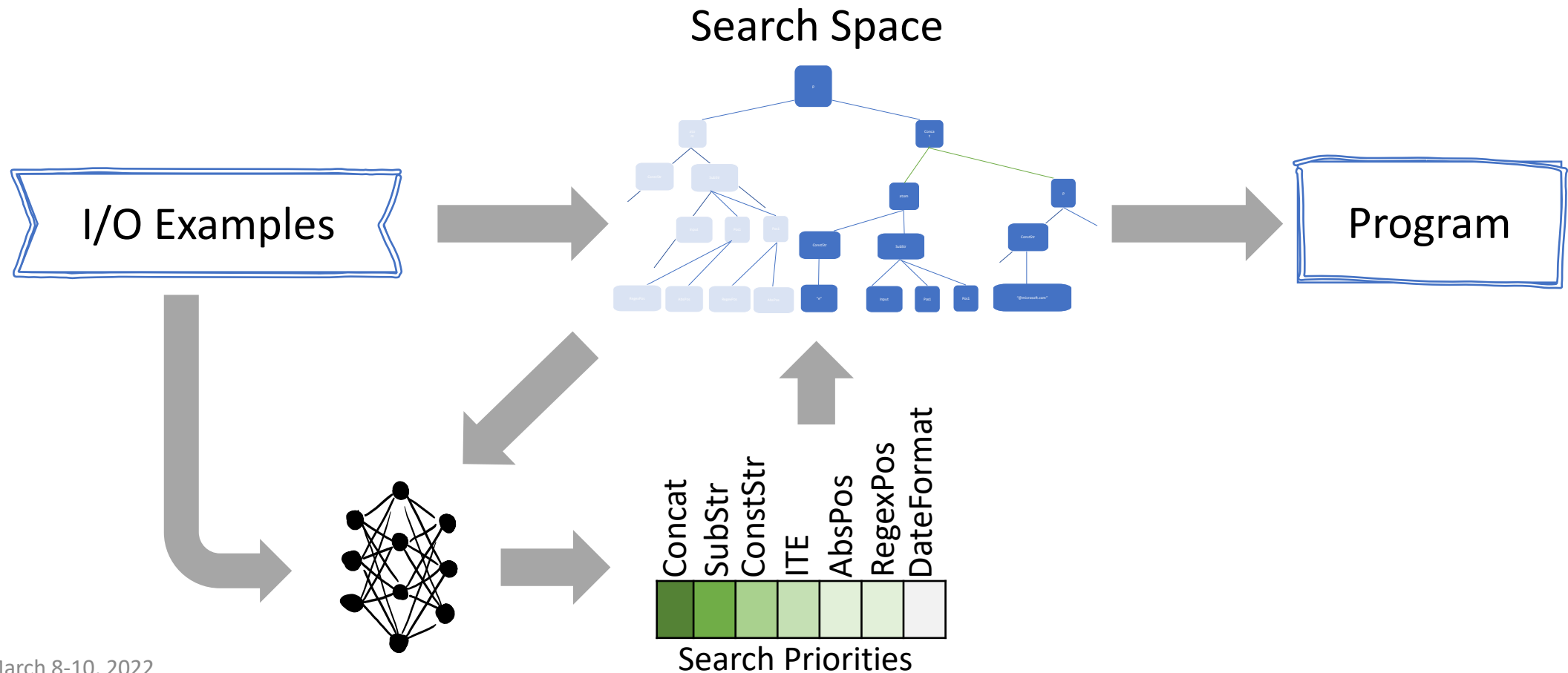
Idea: Order the search space based on a priority list from DNN *before starting*



Neural-Guided Deductive Search

[Kalyan et al., 2018]

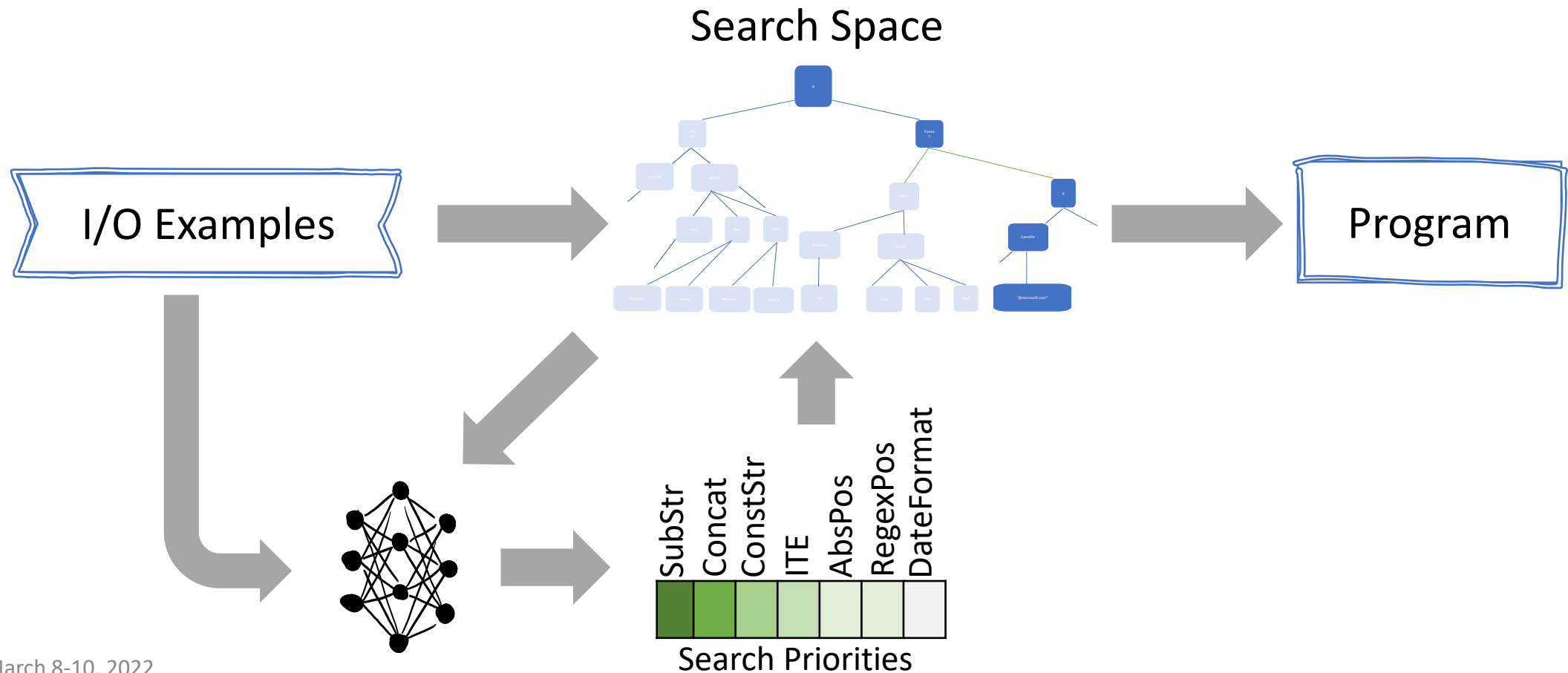
Idea: Order the search space based on a priority list from DNN *at each step*



Neural-Guided Deductive Search

[Kalyan et al., 2018]

Idea: Order the search space based on a priority list from DNN *at each step*



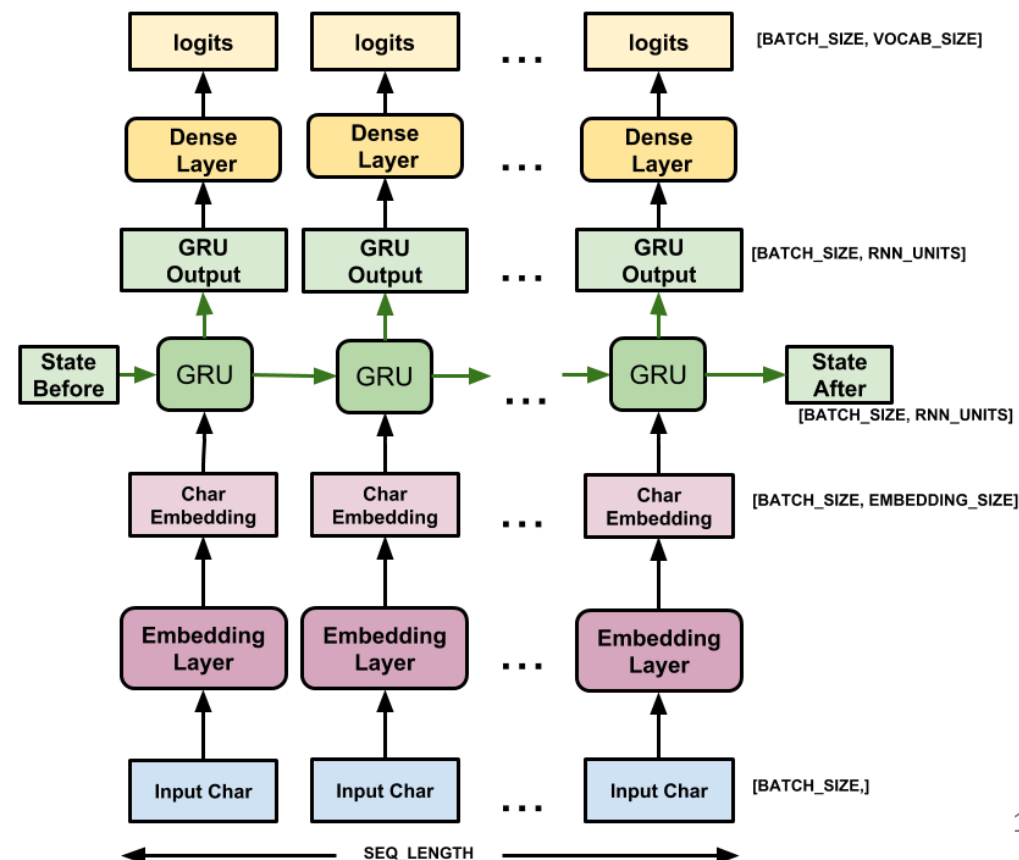
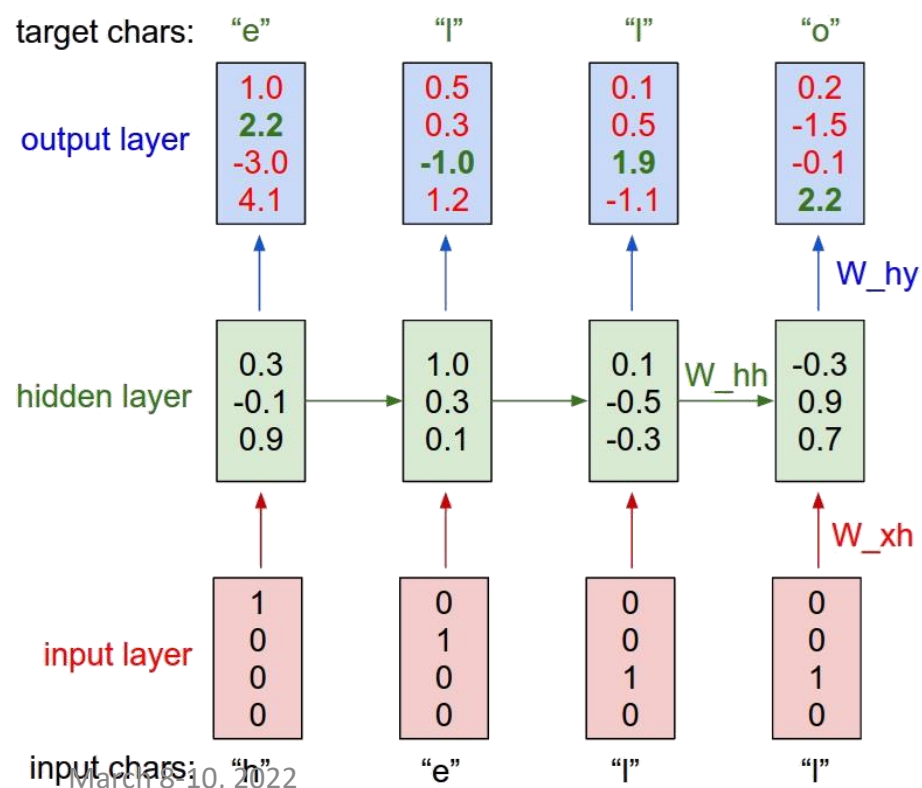
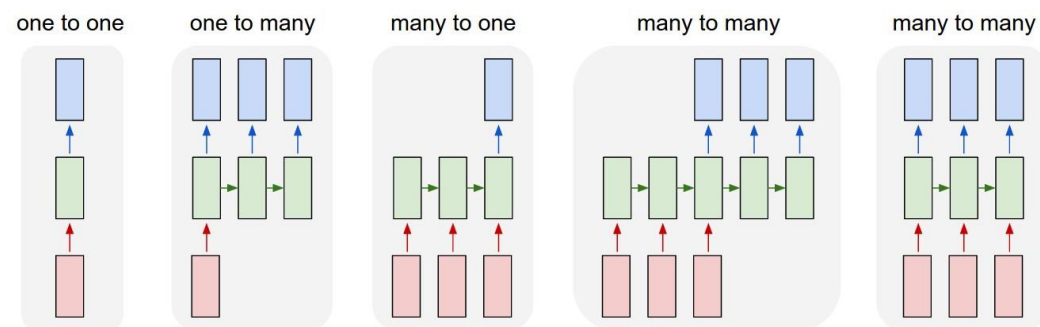
Search branch prediction

- Collect a complete dataset of intermediate search results:

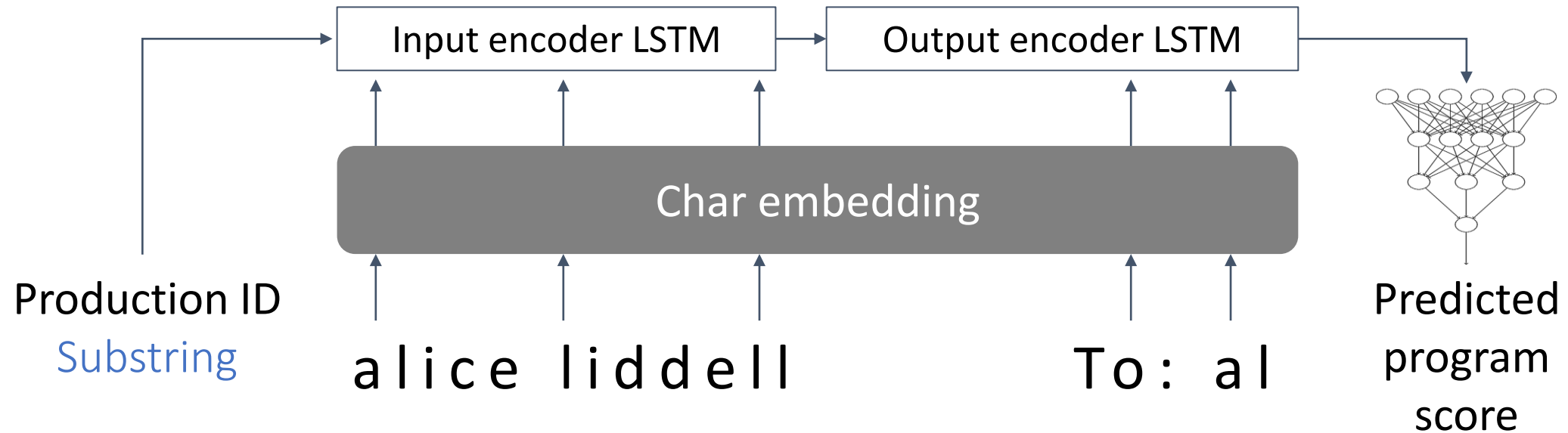
at a search branch $N := F_1(\dots) \mid F_2(\dots) \mid \dots \mid F_k(\dots)$
given a spec $\varphi = \{x \rightsquigarrow y\}$
produced programs P_1, \dots, P_k with scores $h(P_1, \varphi), \dots, h(P_k, \varphi)$

- Learn a predictive model f s.t. $f(F_j, \varphi) \approx h(P_j, \varphi)$
 - φ is an input-output example spec: $\varphi = \{x \mapsto y\}$
 - $f: (\text{enum production_id}, \text{string } x, \text{string } y) \rightarrow \text{float}$

NLP with Recurrent Neural Networks (GRU/LSTM)



Model



- Guides the search toward more likely (higher-quality) programs
- Eliminates unproductive search branches
- Balances performance vs. generalization using branch-and-bound search

Search branch prediction

- Collect a complete dataset of intermediate search results:

at a search branch $N := F_1(\dots) \mid F_2(\dots) \mid \dots \mid F_k(\dots)$
given a spec $\varphi = \{x \rightsquigarrow y\}$
produced programs P_1, \dots, P_k with scores $h(P_1, \varphi), \dots, h(P_k, \varphi)$

- Learn a predictive model f s.t. $f(F_j, \varphi) \approx h(P_j, \varphi)$
- Train using squared-error loss over program scores:

$$\text{Objective: } \mathcal{L}(f; F_j, \varphi) = [f(F_j, \varphi) - h(P_j, \varphi)]^2$$

Q: Why not re-ranking of branches?

- Because the magnitude of score values matters.

Takeaways

- Neural networks excel at noticing patterns in input data
 - “Notice patterns” = *embed* inputs into a lower-dimensional space where related things are close
 - Requires appropriate network architecture, e.g. RNNs for sequential data
- To train a neural network, you need enough data + appropriate loss
 - Rule of thumb: 10-100K diverse data points for an “average” task
- Don’t expect magic, the task must be solvable
- The network’s predictions are used within a larger search design



Likelihood models

Recall: inductive learning

1. Focus on programs that match the observations (i.e., the spec).
2. Pick a program that you are likely looking for.

Symbolic approach:

Use space design to simplify (2).

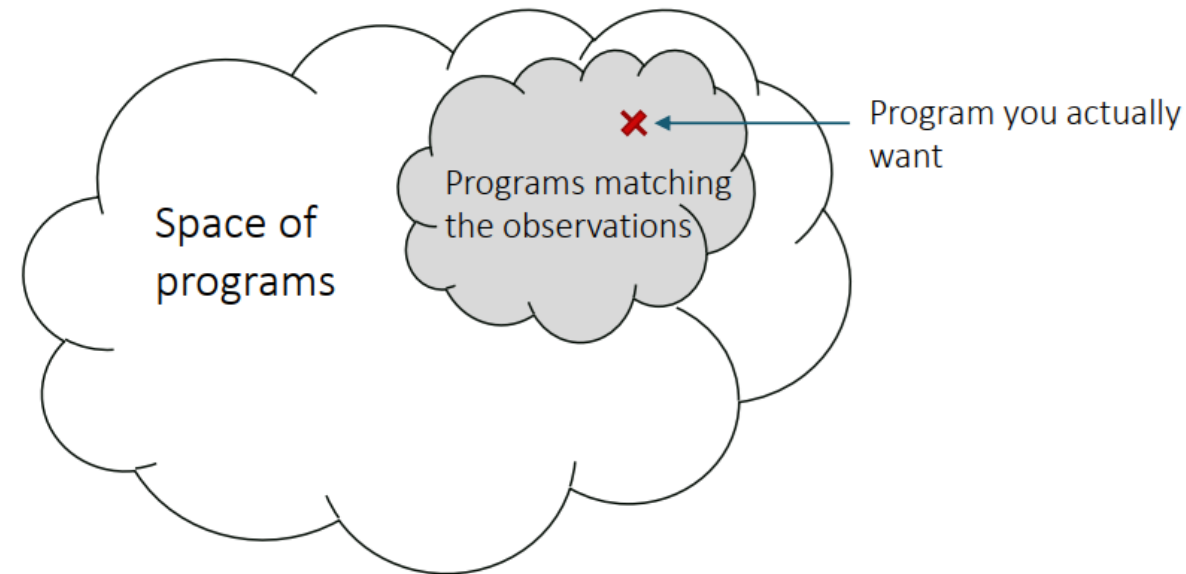
Neural-guided approach:

Make (2) more accurate & faster.

Neural / neuro-symbolic approach:

Relax (1) into a *soft* constraint.

Solve (1) and (2) jointly.



RobustFill: purely-neural synthesis

Devlin, J., Uesato, J., Bhupatiraju, S., Singh, R., Mohamed, A. R., & Kohli, P. RobustFill: Neural program learning under noisy I/O. In *ICML 2017* (pp. 990-998).

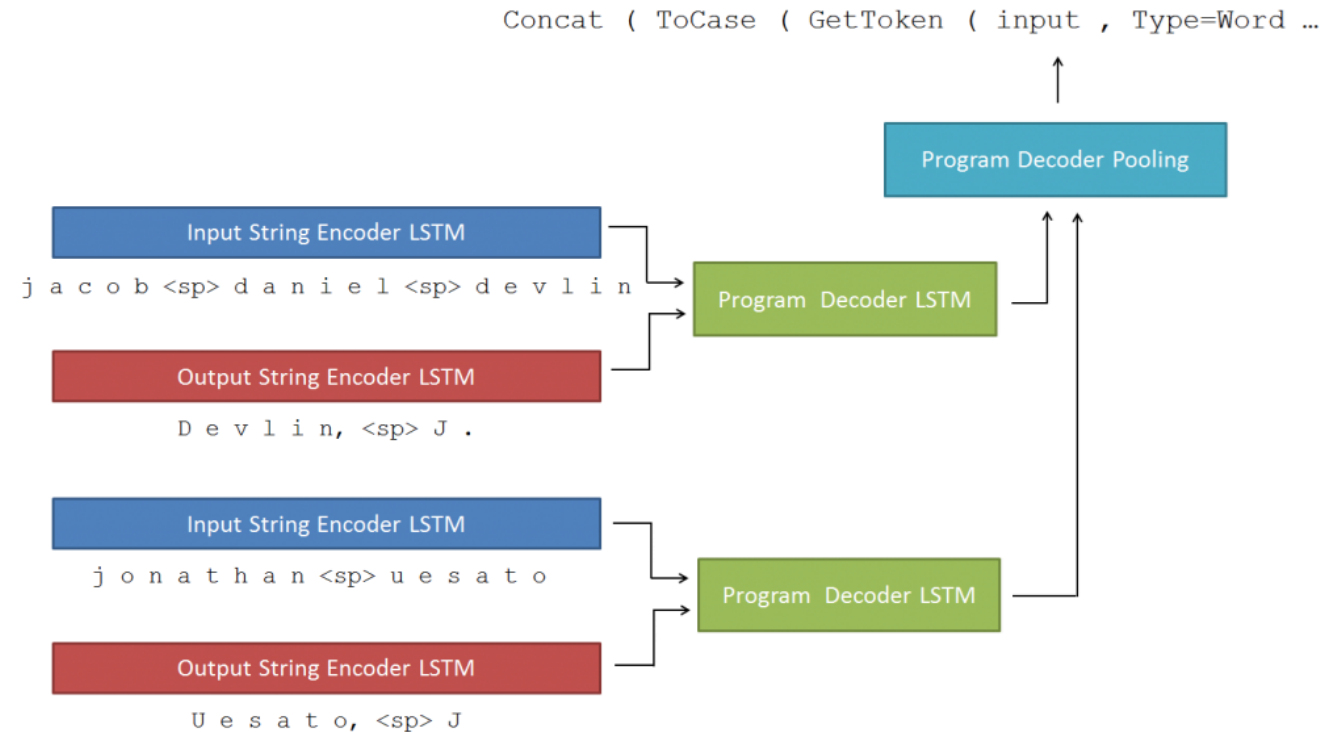
Input String	Output String
jacob daniel devlin	Devlin, J.
jonathan uesato	Useato, J
Surya Bhupatiraju	Bhupatiraju S.
Rishabh q. singh	Singh, R.
abdelrahman mohamed	Mohamed, A.
pushmeet kohli	Kohli, P.

```
Concat(  
  ToCase(  
    GetToken(  
      input,  
      Type=Word,  
      Index=-1),  
    Type=Proper),  
  Const(", "),  
  ToCase(  
    SubString(  
      GetToken(  
        input,  
        Type=Word,  
        Index=1),  
      Start=0,  
      End=1),  
    Type=Proper),  
  Const("."))
```

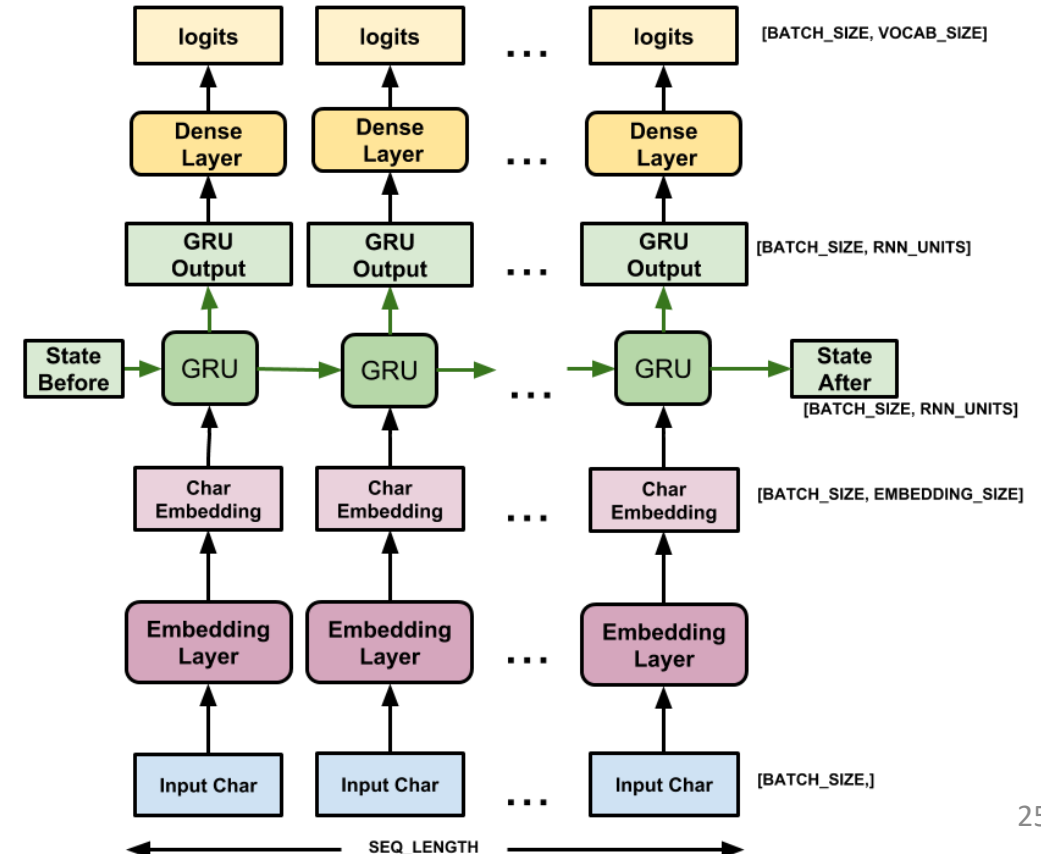
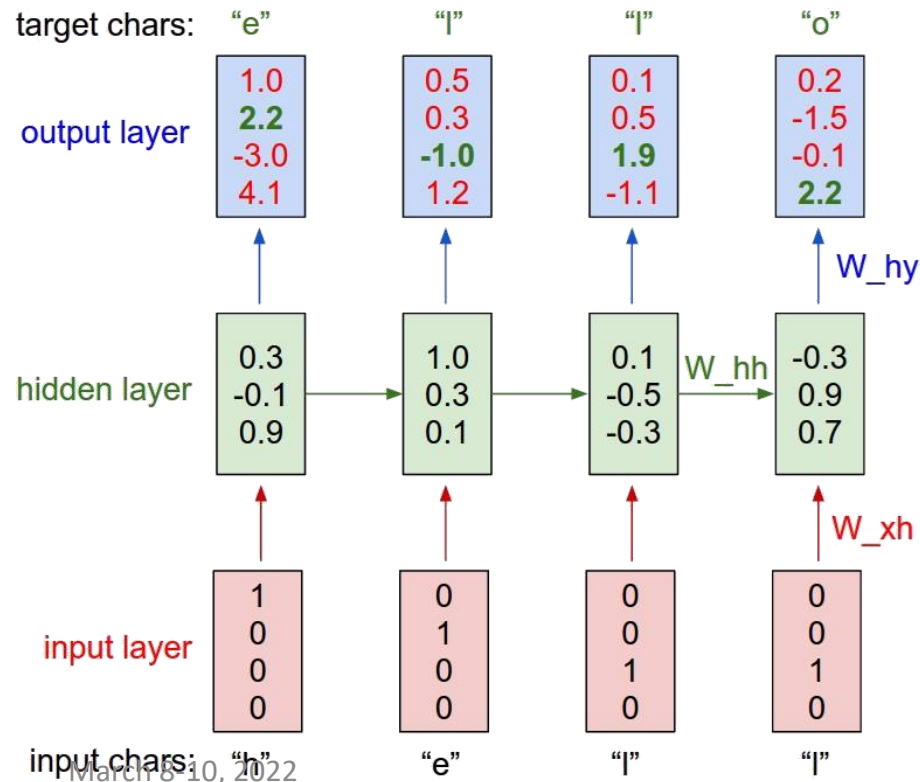
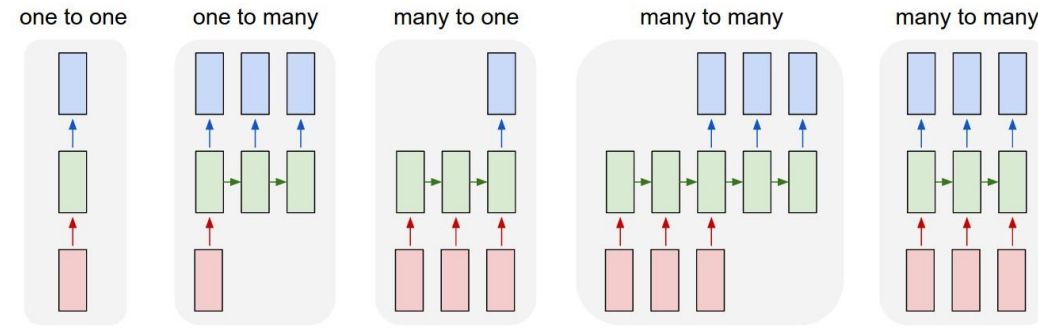
RobustFill: purely-neural synthesis

Key ideas:

- Embed I/O examples with LSTM encoders
- Emit program tokens with LSTM decoders
- Train from large-scale random data



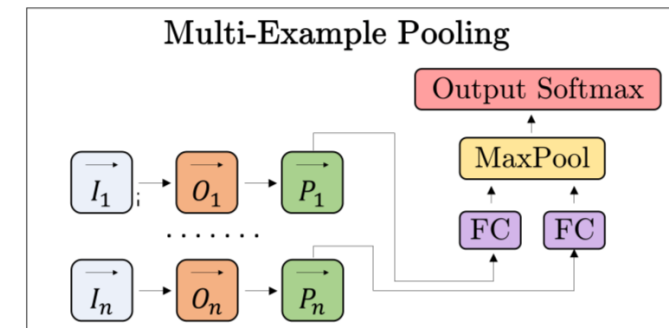
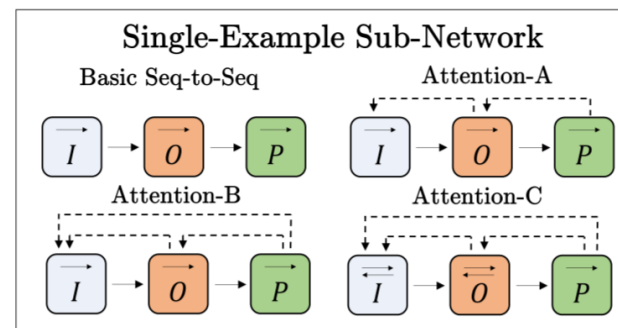
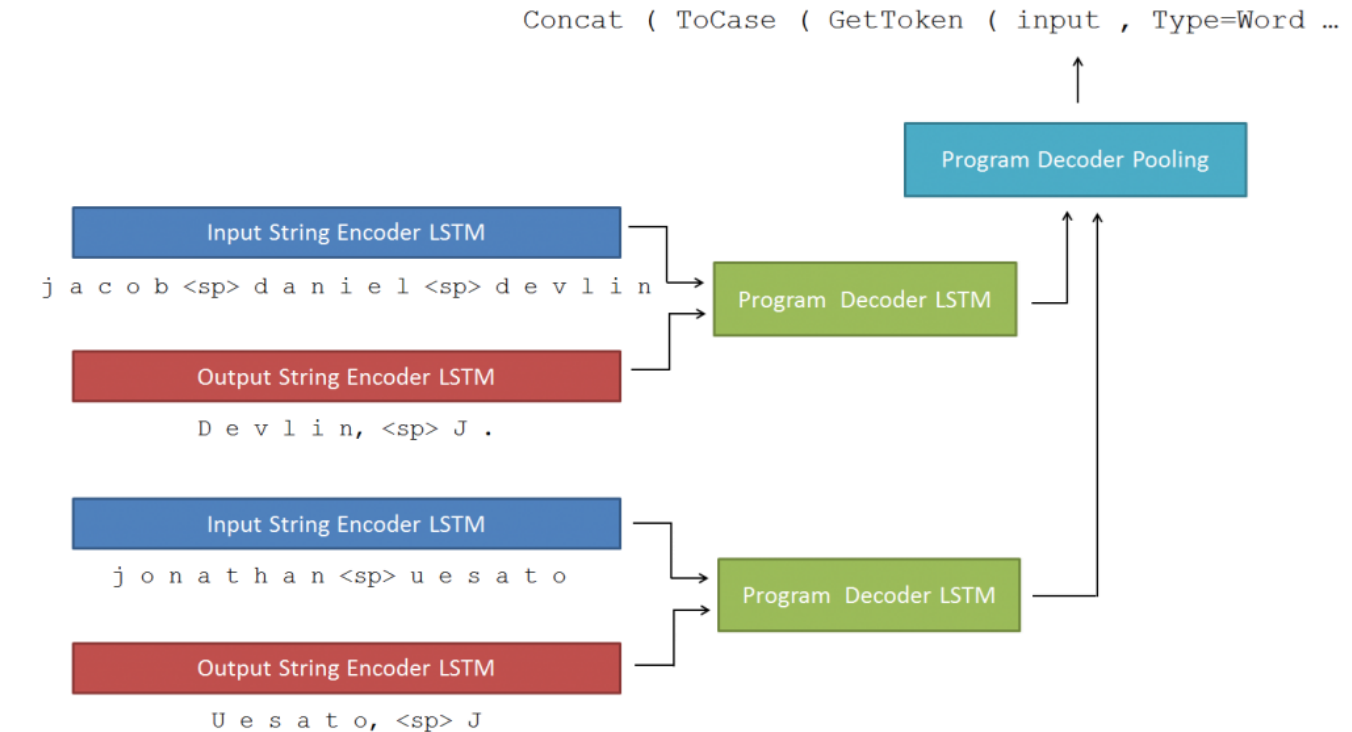
NLP with Recurrent Neural Networks (GRU/LSTM)



RobustFill: purely-neural synthesis

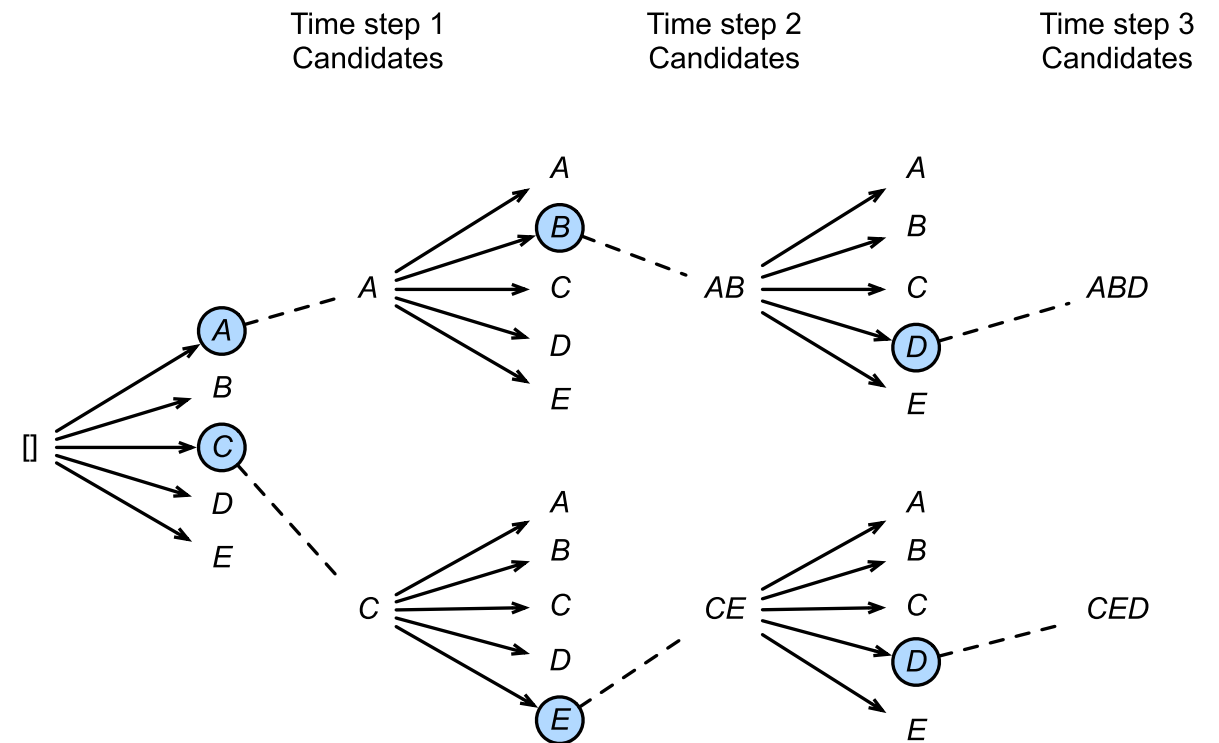
Key ideas:

- Embed I/O examples with LSTM encoders
- Emit program tokens with LSTM decoders
- Train from large-scale random data
- Architecture:
 - *Pooling* across examples at each step to predict one program token
 - *Attention* to examples during program decoding
- Beam search with *execution constraints*
 - Execute decoded subexpressions; remove programs whose outputs are not prefixes of the target



Beam search with constraints

	Neural-Guided Search	Beam Search
Search space at each step	Valid rule expansions in the grammar	Any token from the program vocabulary
Selection	Top-K model predictions in the current search branch	Top-K model predictions across all search branches
Valid syntax?	By design	Not necessarily; requires constraints
Valid semantics?	Partially ensured by top-down propagation	Not necessarily; requires constraints



Source: Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2021). Dive into deep learning.

RobustFill: pros and cons

- + Synthesis + ranking in one model
- + Adapts to any DSL extension without custom propagation logic
- + Noise-tolerant
- Does not *guarantee* consistency with I/O examples (even w/o noise)
- Non-deterministic
- Beam search is exponentially slow
- Hard to design synthetic data generation realistically
- Requires constraints/postprocessing to ensure grammar syntax

Neuro-Symbolic Program Synthesis

Symbolic	Neural	Neuro-Symbolic
Search driven by logical inference & grammar syntax rules	Search driven by statistical likelihood model	
Any program consistent with the spec constraints	Most likely program according to the model	
DSLs or languages with rich semantics	Any language with data	
Exponentially slow	Fast model inference, slow search	

Neuro-Symbolic Program Synthesis

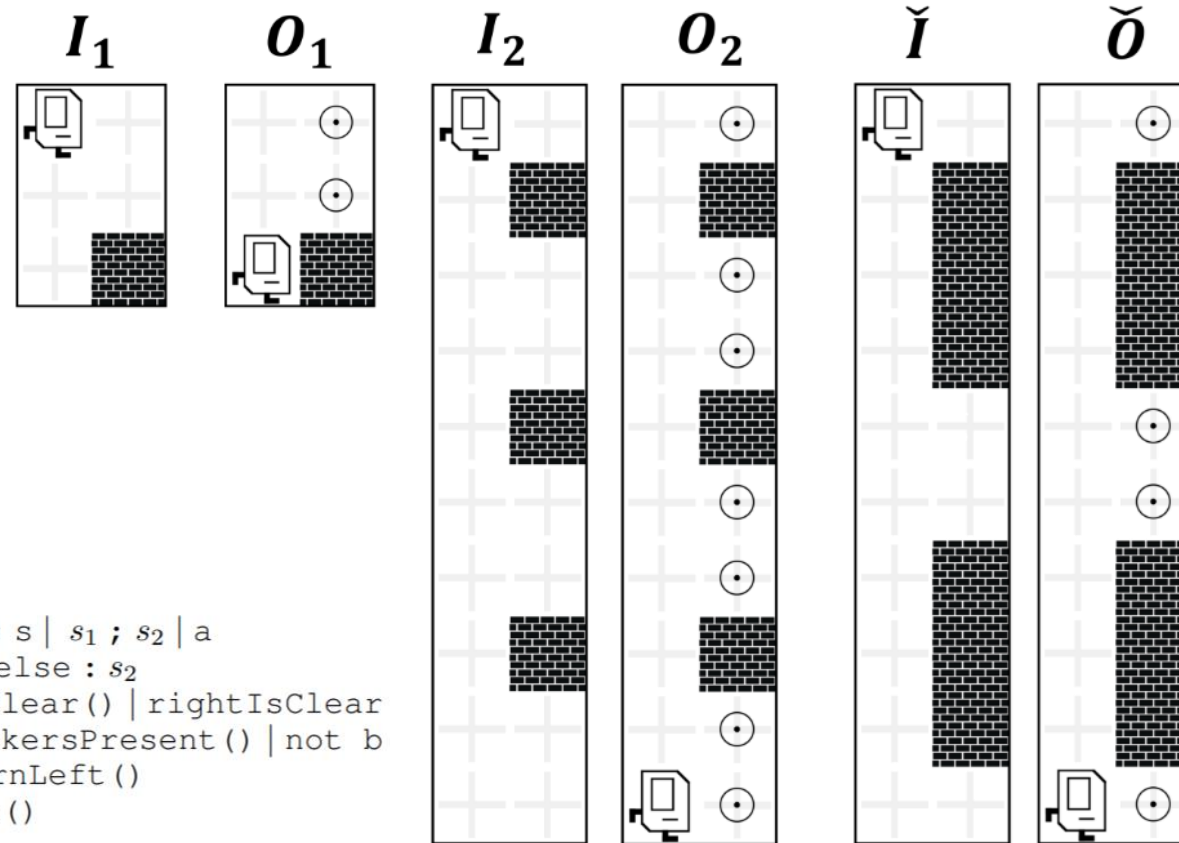
Symbolic	Neural	Neuro-Symbolic
Search driven by logical inference & grammar syntax rules	Search driven by statistical likelihood model	Search driven by likelihood model constrained to only syntactically & logically valid options
Any program consistent with the spec constraints	Most likely program according to the model	Most likely program according to the model, subject to constraints
DSLs or languages with rich semantics	Any language with data	Any language with data, guarantees vary
Exponentially slow	Fast model inference, slow search	Fast model inference, slower search

Execution guidance in the model

- The model only learns from *syntax* of the output programs
- Constrained beam search filters programs via partial execution
 - Expensive, late, wasteful
- How do we teach the model itself what its generated programs do?

Chen, X., Liu, C., & Song, D. (2018). Execution-guided neural program synthesis. In *International Conference on Learning Representations*.

Karel: robot navigation DSL



Underlying Program
(Not used by model)

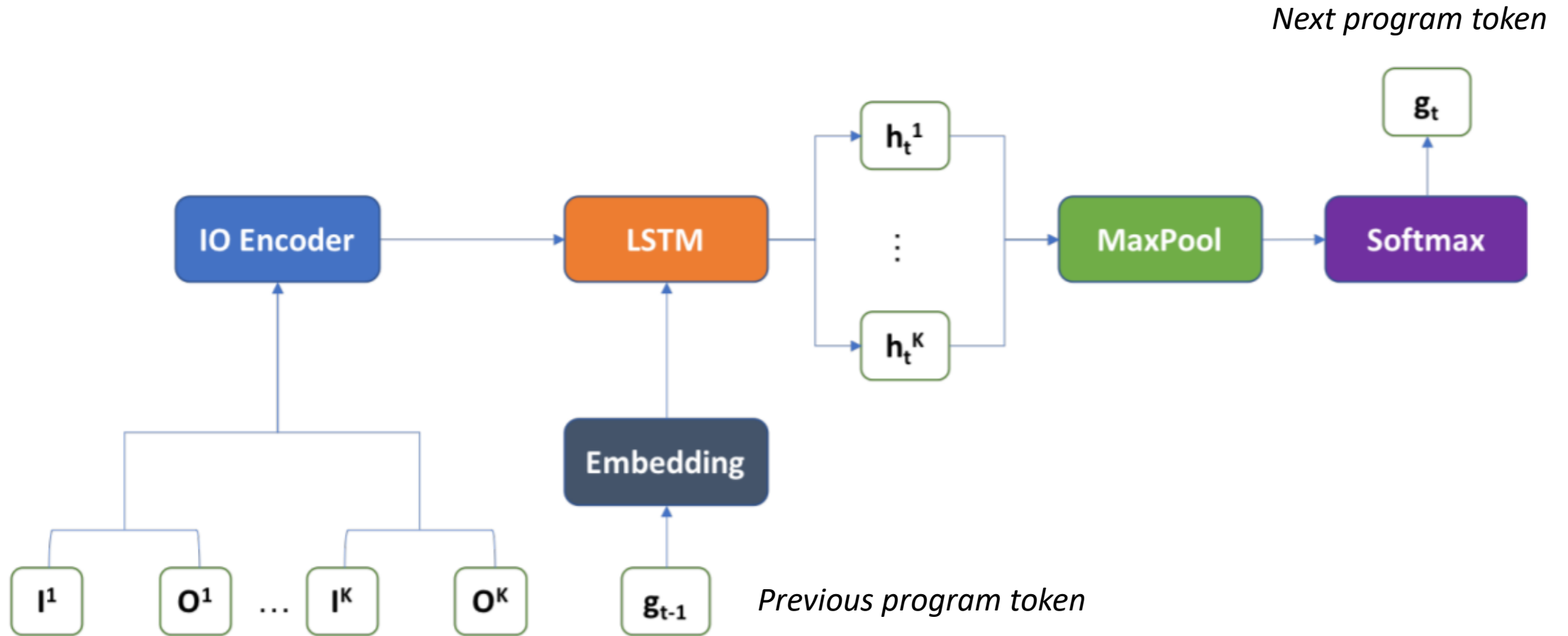
```
def run():
    if rightIsClear():
        turnRight()
        move()
        putMarker()
        turnLeft()
        turnLeft()
        move()
        turnRight()
    while frontIsClear():
        move()
        if rightIsClear():
            turnRight()
            move()
            putMarker()
            turnLeft()
            turnLeft()
            move()
            turnRight()
```

```

Prog p ::= def run() : s
Stmt s ::= while(b) : s | repeat(r) : s | s1 ; s2 | a
        | if(b) : s | ifelse(b) : s1 else : s2
Cond b ::= frontIsClear() | leftIsClear() | rightIsClear
        | markersPresent() | noMarkersPresent() | not b
Action a ::= move() | turnRight() | turnLeft()
          | pickMarker() | putMarker()
Cste r ::= 0 | 1 | ... | 19

```


Base model for neural synthesis



Discussion

Suggestive definitions. In the first avenue, a new technical term is coined that has a suggestive colloquial meaning, thus sneaking in connotations without the need to argue for them. This often manifests in anthropomorphic characterizations of tasks (*reading comprehension* and *music composition*) and techniques (*curiosity* and *fear*). I (Zachary) am responsible for the latter). A number of papers name components of proposed models in a manner suggestive of human cognition (for example, *thought vectors* and the *consciousness prior*). Our goal is not to rid the academic literature of all such language; when properly qualified, these connections might communicate a fruitful source of inspiration. When a suggestive term is assigned technical meaning, however, each subsequent paper has no choice but to confuse its readers, either by embracing the term or by replacing it.

1. What does the synthesis model “understand” about semantics of the program behavior on the examples?
2. How does the paper improve the model’s understanding?
3. What’s changed outside of the model?
4. Do we ensure consistency, maximize likelihood, or both?
5. What’s the difference between this and RobustFill’s constrained beam search approach?
6. What are the issues with loops and conditionals?
7. Where is this applicable?

Informal specifications

Specs in real-world program synthesis

- Until now, this class has dealt with *formal* specs
 - i.e., given a program P and a spec φ , whether $P \models \varphi$ is objectively verifiable
- However, most practical specs are informal
 - Natural language
 - Sketches / diagrams
 - Context
- Moreover, some specs are formal yet hard to reason about
 - Karel's maze maps are big and complex – SMT solvers do not scale

Example: NL→SQL

Wang, B., Shin, R., Liu, X., Polozov, O., & Richardson, M. (2020, July). RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers. In *Proceedings of the ACL* (pp. 7567-7578).

Database Question Answering

about

employee_hire_evaluation

7/16 12:39 PM

Which shop sells the largest number of products? List the manager name and district.

DBQA 7/16 12:39 PM

Tommi Kautonen, Arto Tolsa Areena

7/16 12:39 PM

Which shop's number products is below the average? Give me the shop names.

DBQA 7/16 12:39 PM

FC Haka; FC Honka; FF Jaro; FC KooTeePee; KuPS; IFK Mariehamn

Type your question here...



... or click here to choose an example question.

Output (4)

Input (4)

"employee"

Primary Key: "Employee_ID"

Employee_ID	Name	Age	City
1	George Chuter	23	Bristol
2	Lee Mears	29	Bath
3	Mark Regan	43	Bristol
4	Jason Hobson	30	Bristol
5	Tim Payne	29	Wasps
6	Andrew Sheridan	28	Sale
7	Matt Stevens	29	Bath
8	Phil Vickery	40	Wasps
9	Steve Borthwick	32	Bath
10	Louis Deacon	36	Leicester

10 rows, 4 columns - you can scroll the content. Columns are sortable - click on a header to sort.

"shop"

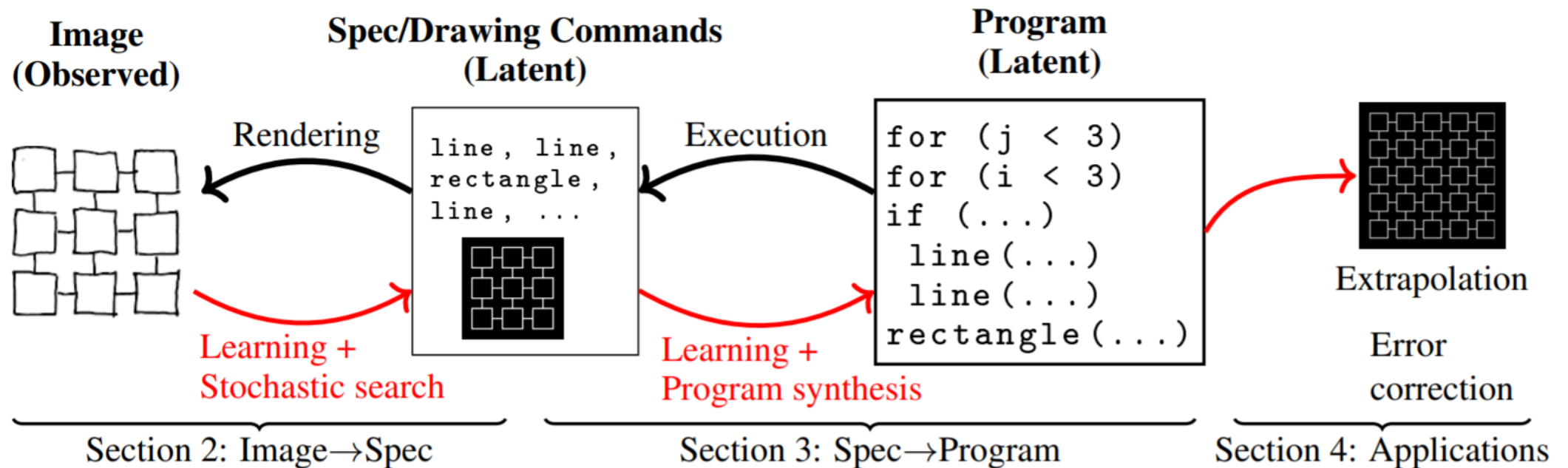
Primary Key: "Shop_ID"

Shop_ID	Name	Location	District	Number_products	Manager_name
1	FC Haka	Valkeakoski	Tehtaan kenttä	3516	Olli Huttunen
2	HJK	Helsinki	Finnair Stadium	10770	Antti Muurinen
3	FC Honka	Espoo	Tapiolan Urheilupuisto	6000	Mika Lehtosuo
4	FC Inter	Turku	Veritas Stadion	10000	Job Dragtsma
5	FF Jaro	Jakobstad	Jakobstads Centralplan	5000	Mika Laurikainen
6	FC KooTeePee	Kotka	Arto Tolsa Areena	4780	Tommi Kautonen
7	KuPS	Kuopio	Magnum Areena	3500	Kai Nyysönen
8	FC Lahti	Lahti	Lahden Stadion	15000	Ilkka Mäkelä
9	IFK Mariehamn	Mariehamn	Wiklöf Holding Arena	1600	Pekka Lyyski

9 rows, 6 columns - you can scroll the content. Columns are sortable - click on a header to sort.

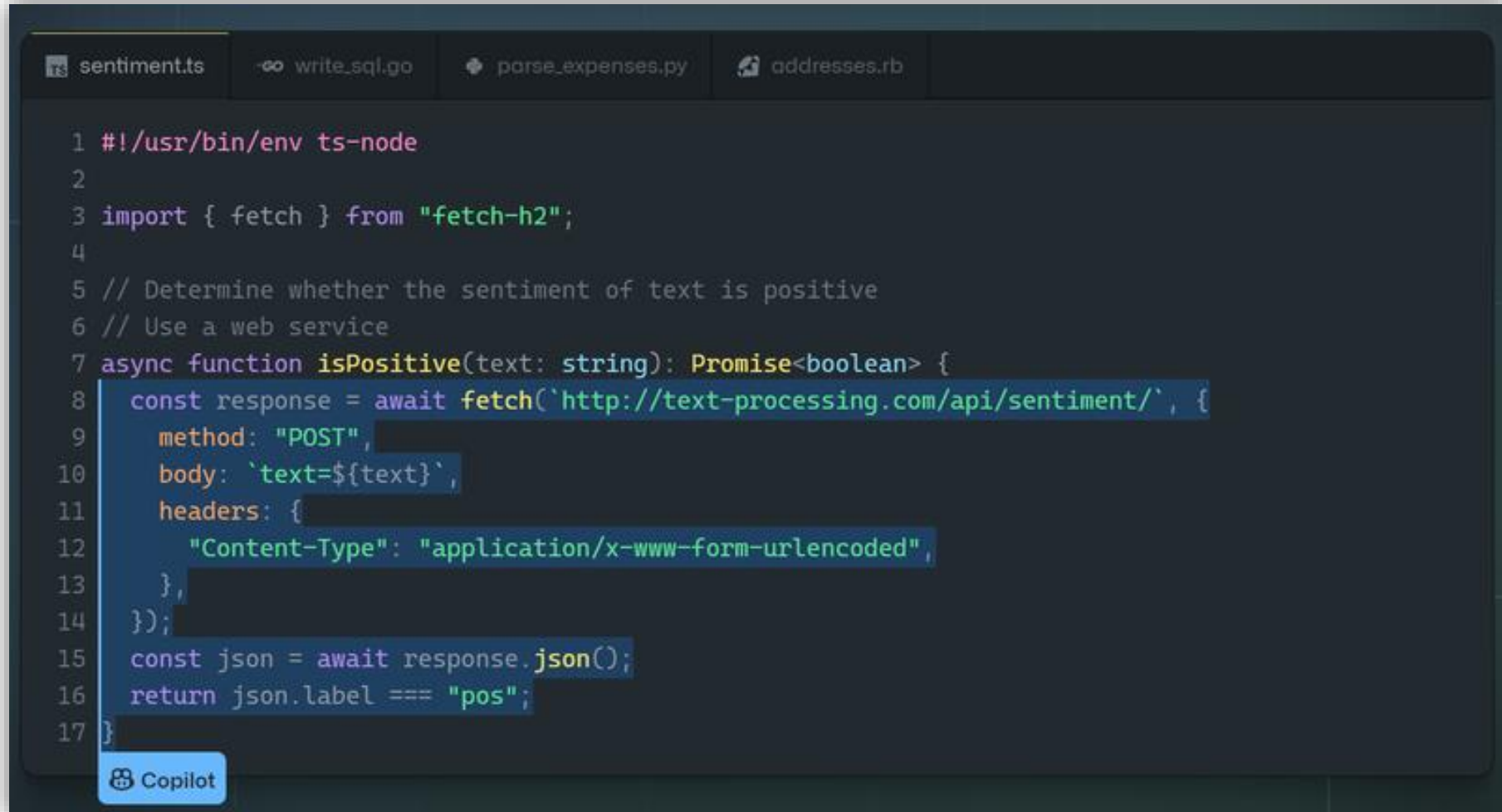
Example: Diagram generation

Ellis, K., Ritchie, D., Solar-Lezama, A., & Tenenbaum, J. (2018). Learning to infer graphics programs from hand-drawn images. *NeurIPS*, 31.



Example: Code completion

Source: Github Copilot



The screenshot shows a code editor with four tabs: `sentiment.ts`, `write_sql.go`, `parse_expenses.py`, and `addresses.rb`. The active tab is `sentiment.ts`, which contains the following TypeScript code:

```
1 #!/usr/bin/env ts-node
2
3 import { fetch } from "fetch-h2";
4
5 // Determine whether the sentiment of text is positive
6 // Use a web service
7 async function isPositive(text: string): Promise<boolean> {
8   const response = await fetch('http://text-processing.com/api/sentiment/', {
9     method: "POST",
10    body: `text=${text}`,
11    headers: {
12      "Content-Type": "application/x-www-form-urlencoded",
13    },
14  });
15  const json = await response.json();
16  return json.label === "pos";
17 }
```

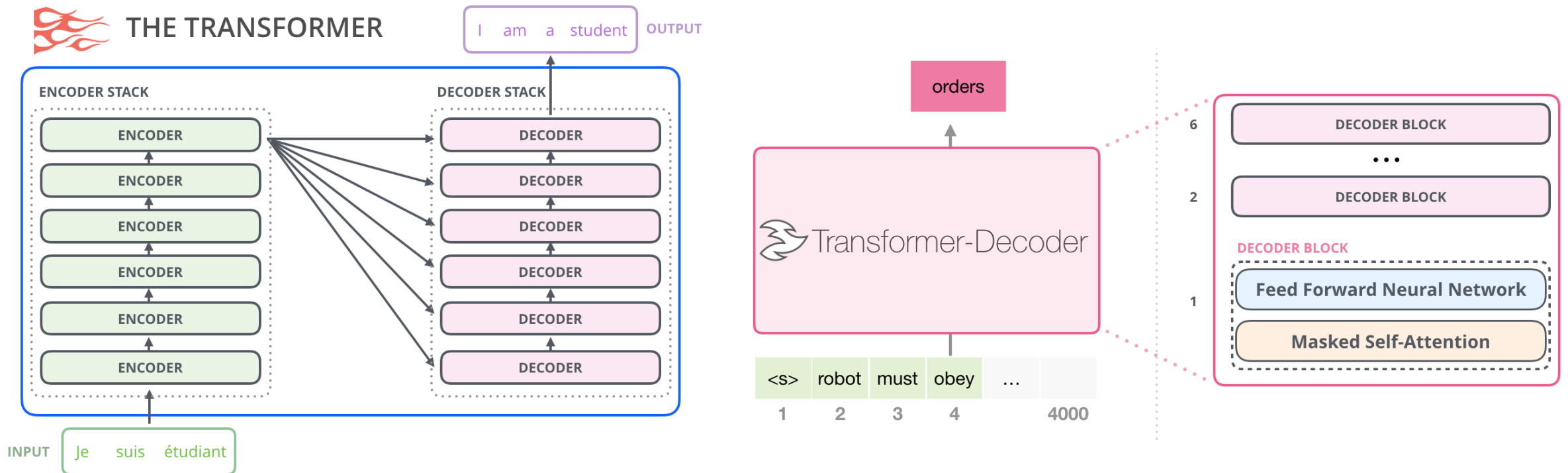
GitHub Copilot suggestions are shown as blue highlights on lines 8 through 14, covering the `fetch` call and the `headers` object. A Copilot logo is visible in the bottom-left corner of the editor.

If $P \models \varphi$ is unverifiable, how do we know P is good?

- We don't. Not for sure.
- As always in neural program synthesis, the goal is $\text{argmax Pr}[P \mid \varphi]$.
 - Synthesis + ranking in one model
- But we can improve the likelihood of valid programs:
 - Instead of token-by-token RNNs, use models that ensure grammar syntax
 - During beam search, eliminate programs that don't compile, type-check, cause runtime errors...
 - Add relevant symbolic information into the context φ
 - Train program verifiers
 - Facilitate iteration and clarification

Let's start with a base model

Github Copilot (really OpenAI Codex (really GPT LM)) from 1000 ft:



Case study #1: NL→SQL

City:

id	name	country code	district	pop
----	------	--------------	----------	-----

Country_Language:

country_code	language	is official	percent
--------------	----------	-------------	---------

Country:

code	name	continent	capital	life_expectancy	government form	...
------	------	-----------	---------	-----------------	-----------------	-----

What is the average life expectancy in the countries where English is not the official language?

```
SELECT AVG(life_expectancy)
FROM country
WHERE name NOT IN
    (SELECT T1.name
     FROM country AS T1 JOIN
     country_language AS T2
     ON T1.code = T2.country_code
     WHERE T2.language = "English"
     AND T2.is_official = "T")
```

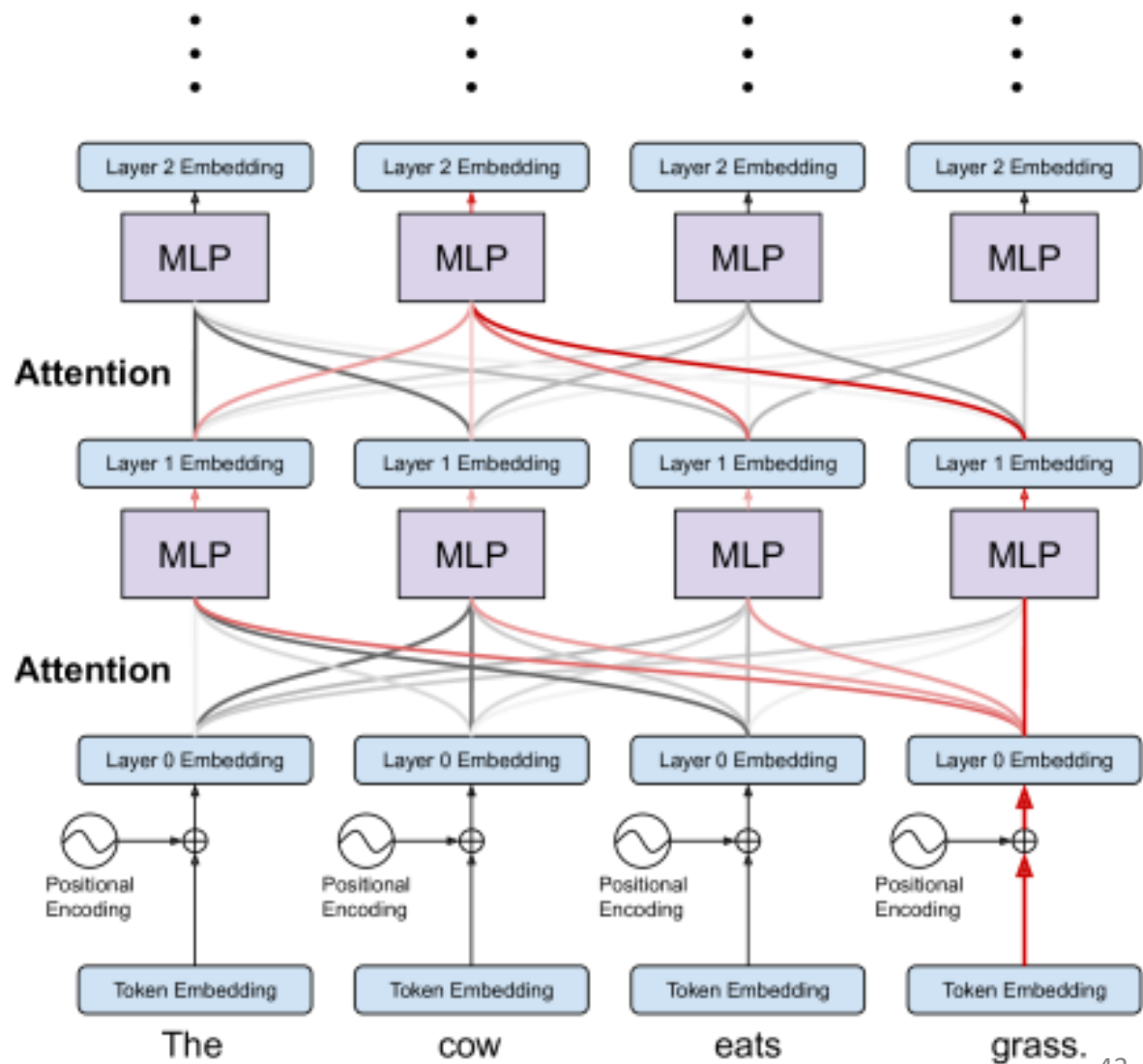
Transformer

[Vaswani et al., NeurIPS 2017]

$$\mathbf{x}_i \rightsquigarrow \mathbf{q}_i, \mathbf{k}_i, \mathbf{v}_i$$

$$\alpha_{ij} = \text{softmax}_j \frac{\mathbf{q}_i \mathbf{k}_j^\top}{\sqrt{\text{dim}}}$$

$$\mathbf{y}_i = \sum_j \alpha_{ij} \mathbf{v}_j$$



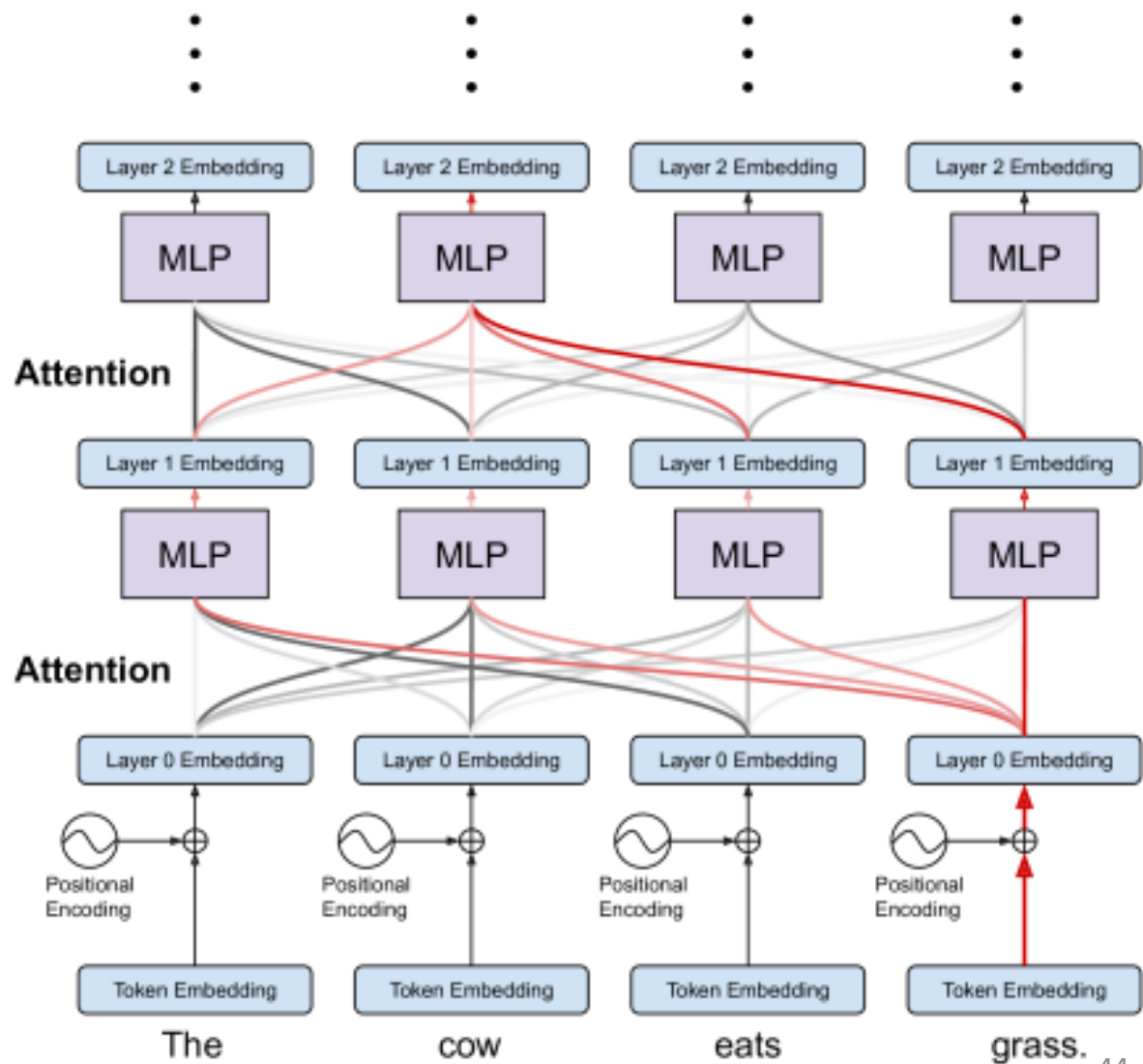
Transformer

[Vaswani et al., NeurIPS 2017]

$$\mathbf{x}_i \rightsquigarrow \mathbf{q}_i, \mathbf{k}_i, \mathbf{v}_i$$

$$\alpha_{ij} = \text{softmax}_j \frac{\mathbf{q}_i \mathbf{k}_j^\top}{\sqrt{\text{dim}}}$$

$$\mathbf{y}_i = \sum_j \alpha_{ij} \mathbf{v}_j$$



Relation-Aware Transformer (RAT)

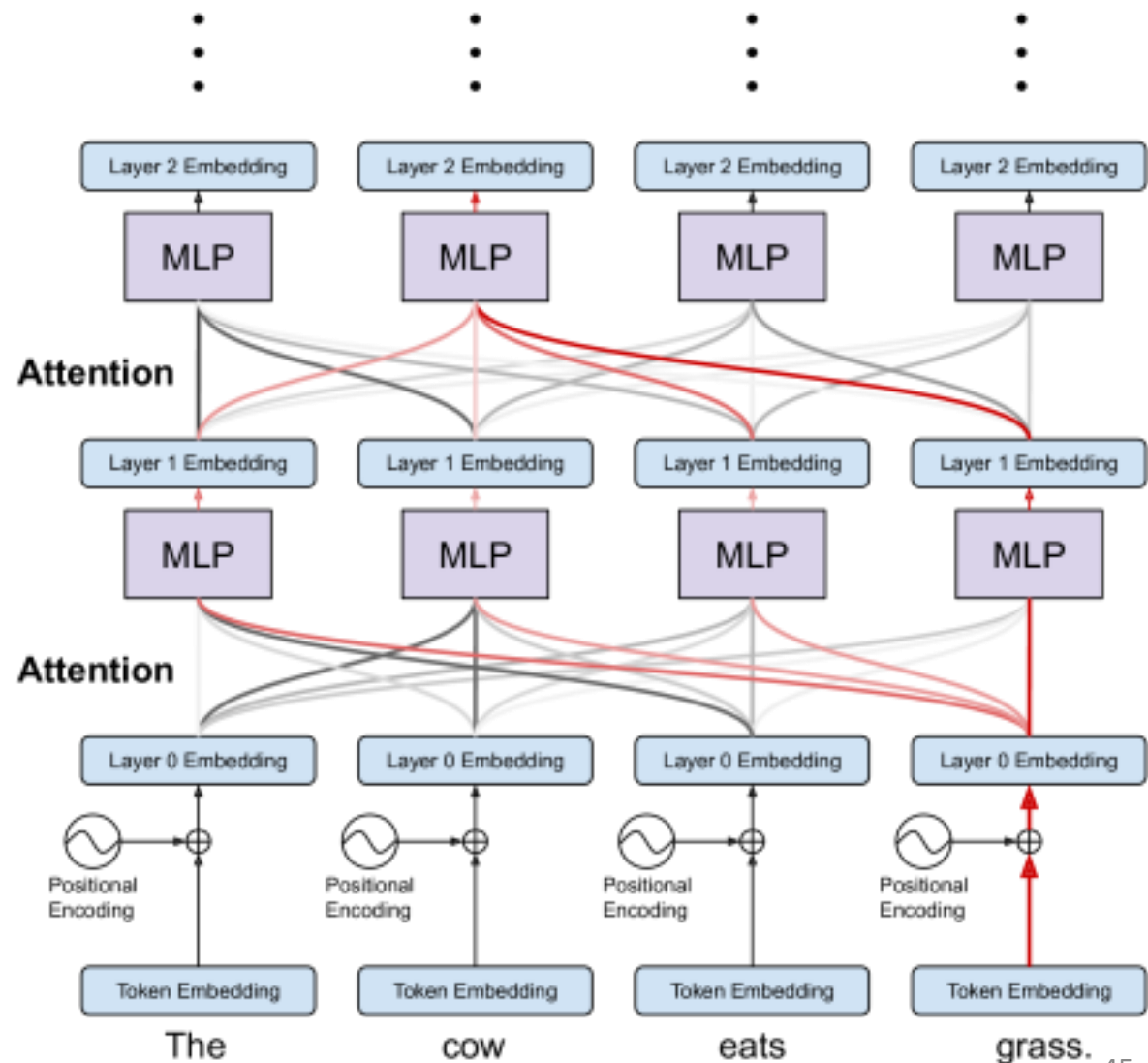
[Vaswani et al., NeurIPS 2017] [Shaw et al., NAACL 2018]

$$\mathbf{x}_i \rightsquigarrow \mathbf{q}_i, \mathbf{k}_i, \mathbf{v}_i$$

$$\alpha_{ij} = \text{softmax}_j \frac{\mathbf{q}_i (\mathbf{k}_j + \boldsymbol{\beta}_{ij})^\top}{\sqrt{\text{dim}}}$$

$$\mathbf{y}_i = \sum_j \alpha_{ij} (\mathbf{v}_j + \boldsymbol{\varepsilon}_{ij})$$

Relative positional embeddings



Relation-Aware Transformer (RAT)

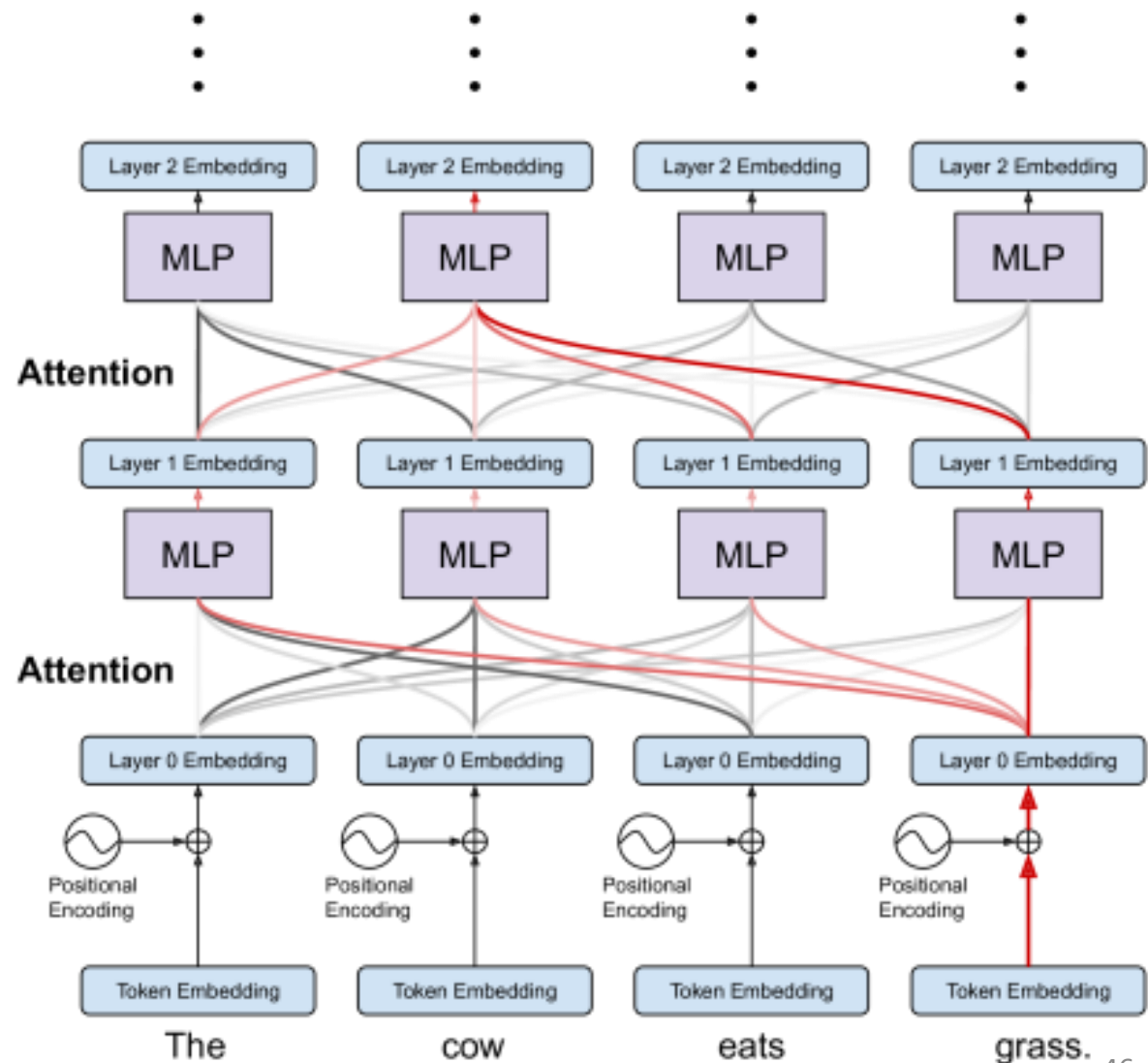
[Vaswani et al., NeurIPS 2017] [Shaw et al., NAACL 2018]

$$\mathbf{x}_i \rightsquigarrow \mathbf{q}_i, \mathbf{k}_i, \mathbf{v}_i$$

$$\alpha_{ij} = \text{softmax}_j \frac{\mathbf{q}_i (\mathbf{k}_j + \boldsymbol{\beta}_{ij})^\top}{\sqrt{\text{dim}}}$$

$$\mathbf{y}_i = \sum_j \alpha_{ij} (\mathbf{v}_j + \boldsymbol{\varepsilon}_{ij})$$

~~Relative positional embeddings~~
Arbitrary edge features



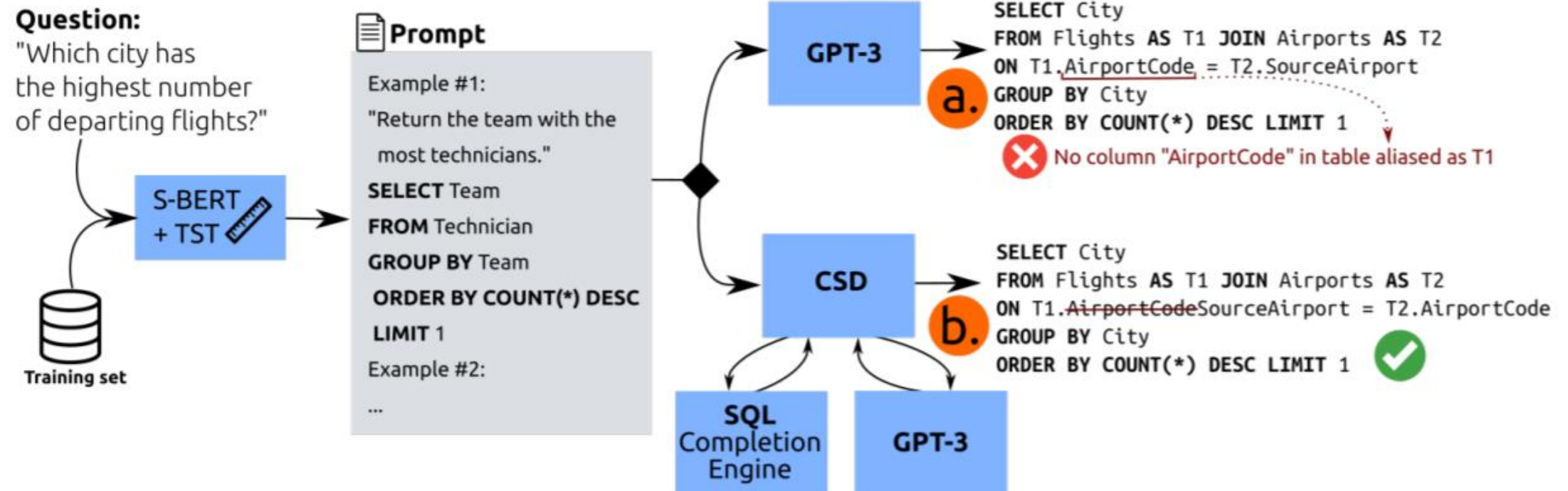
RAT-SQL schema encoding/linking

[Wang and Shin et al., ACL 2020]

1. Foreign key relations.
2. Column/table correspondence.
3. Table/primary key correspondence.
4. Full and partial question word/span matches.
5. Value-based linking (entries that occur in the columns).

One unified *soft* encoding mechanism for **symbolic relational** data.

Case study #2, PL: Synchromesh



Case study #2, PL: Synchromesh

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 ✗
Vega-Lite	Data fields must be used with types compatible with their values.	{"x": {"field": "Category", "type": ^	"nominal" ✓ "temporal" ✗
	Do not facet on field with too many distinct values (breaks rendering).	{"column":{"field": ^	"Category" ✓ "ZipCode" ✗
SMCalFlow	Type-check parameters of all API functions.	(Yield (PlaceHasFeature (^	Takeout ✓ IsWindy ✗ List.Apply ✗
	Track declared variables and their types.	(let (x 85) (Yield (inCelsius ^	x ✓ y ✗

Case study #3, HCl: iteration

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., ... & Sutton, C. (2021). Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

I need to write a Python function called `count_common`. Here's a description: 'Write a function to count the most common words in a dictionary.'

Can you give me an example?

`assert count_common(['one', 'two', 'three', 'four', 'five', 'one', 'two', 'one', 'three', 'one']) == [('one', 4), ('two', 2), ('three', 2), ('four', 1)]`

```
def count_common(list_of_words):
    dict = {}
    for word in list_of_words:
        if word in dict:
            dict[word] += 1
        else:
            dict[word] = 1
    return sorted(dict.items(), key=lambda x: x[1])
```



Close! But can you sort the output in reverse (descending) order?

```
def count_common(list_of_words):
    dict = {}
    for word in list_of_words:
        if word in dict:
            dict[word] += 1
        else:
            dict[word] = 1
    return sorted(dict.items(), key=lambda x: x[1], reverse=True)
```



Great. Now notice that the example only returns the top 4 results.

```
def count_common(list_of_words):
    dict = {}
    for word in list_of_words:
        if word in dict:
            dict[word] += 1
        else:
            dict[word] = 1
    return sorted(dict.items(), key=lambda x: x[1], reverse=True)[:4]
```



Hooray! 🎉🎉🎉

I need to write a Python function called `find_Max_Num`. Here's a description: 'Write a python function to find the largest number that can be formed with the given list of digits.'

Can you give me an example?

`assert find_Max_Num([1,2,3]) == 321`

```
def find_Max_Num(test_list):
    largest = 0
    for i in range(0, len(test_list)):
        current = test_list[i]
        for j in range(0, len(test_list)):
            if (current == test_list[j]):
                current = current * 10
            else:
                break
        if (largest < current):
            largest = current
    return largest
```



This function should sort the list in decreasing order and then combine the digits into an integer.

```
def find_Max_Num(test_list):
    return sorted(test_list)[-1]
```



Join the string representations of the values in the sorted list.

```
def find_Max_Num(test_list):
    return ".join(map(str, sorted(test_list)))
```



Sort the list from highest to lowest, not lowest to highest.

```
def find_Max_Num(test_list):
    return ".join(map(str, sorted(test_list, reverse=True)))
```



Return the int of the string.

```
def find_Max_Num(test_list):
    return int(".join(map(str, sorted(test_list, reverse=True))))
```



Summary

- Symbolic techniques enforce correctness & semantic structure
Neural techniques learn statistical patterns + handle informal specs
- Active research in neuro-symbolic techniques – incorporating symbolic info and correctness guarantees into neural models
- User-facing scenarios broaden the definition of “specs”
Novel PL/ML/HCI approaches to bring back correctness a posteriori
- Pretrained language models enable modern application scenarios