Neural-Guided Deductive Search for Real-Time Program Synthesis from Examples

Ashwin Kalyan ashwinkv@gatech.edu
Abhishek Mohta t-abmoht@microsoft.com
Alex Polozov polozov@microsoft.com
Dhruv Batra dbatra@gatech.edu
Prateek Jain prajain@microsoft.com
Sumit Gulwani sumitg@microsoft.com

Microsoft · Georgia Tech · Microsoft Research AI

Problem

- Program synthesis from input-output examples
- Example domain: string → string transformations for data wrangling

Input	Output
Yoshua Bengio	Y. B.
Hugo Larochelle	
Tara Sainath	
Yann LeCun	

```
Concat( Substring(0, 1),
   Concat( ConstStr(". "),
      Concat(
         Substring(2nd uppercase, +1),
         ConstStr("."))))
```

- Domain-specific language (excerpt):

```
string transform := atom | Concat(atom, transform);
string atom := ConstStr(s)   | Substring(pos1, pos2);
int pos :=   AbsolutePosition(k)
           | RegexPosition(r, k)
           | RelativeOffsetPosition(k);
string s;     int k;     Regex r;
```

- Desiderata: *correct*, *generalizable*, and *fast*

Approach

Key idea: deductive search guided by an offline-trained neural model.

Unify the strengths of *symbolic* and *neural* program synthesis:

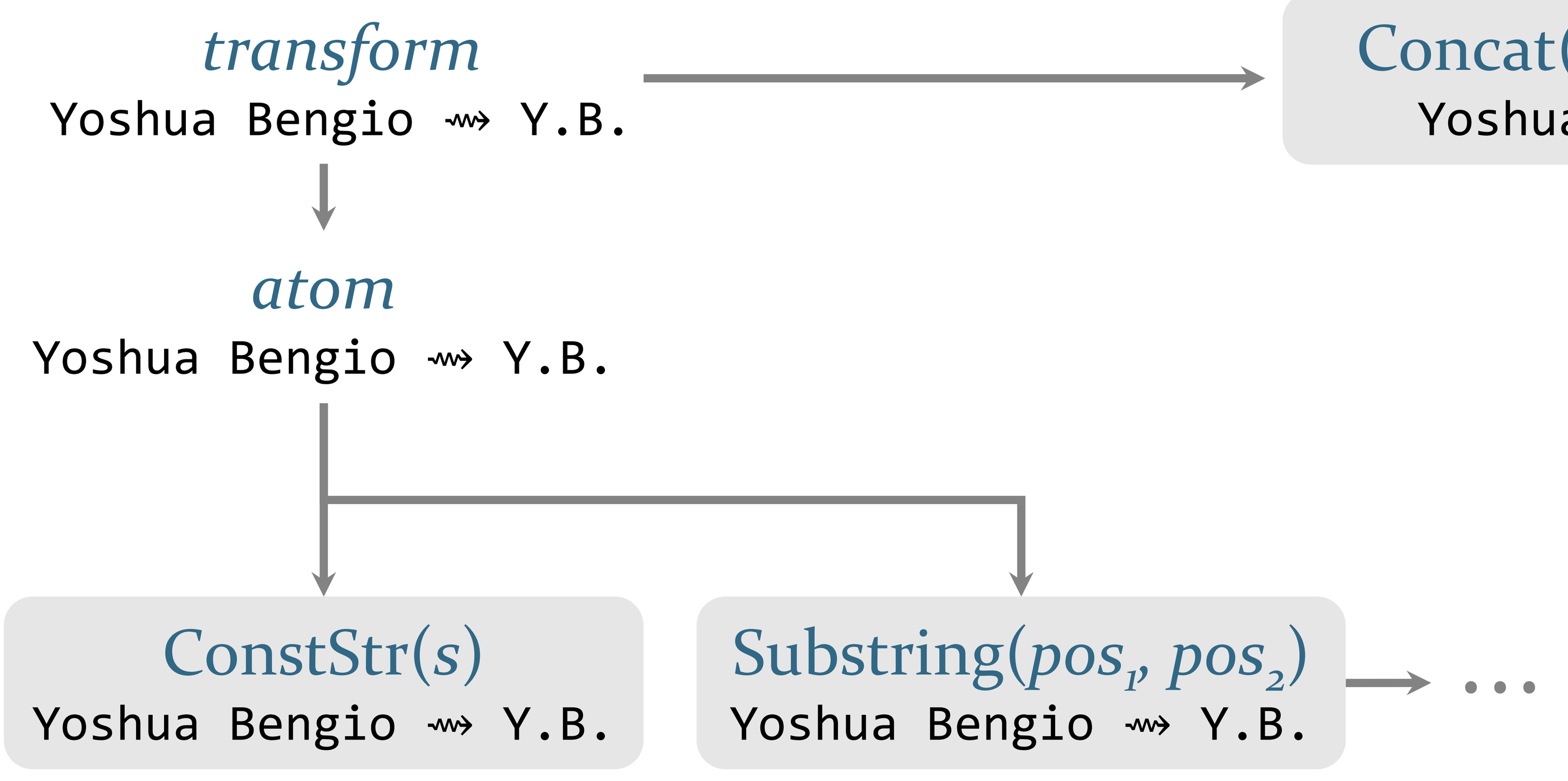
- ✓ Produces programs that are correct by construction
- ✓ Generalizes from a single example in 68% cases
- ✓ Real-time performance of < 0.1 sec
- ✓ Trainable with SGD ⇒ avoids engineering effort

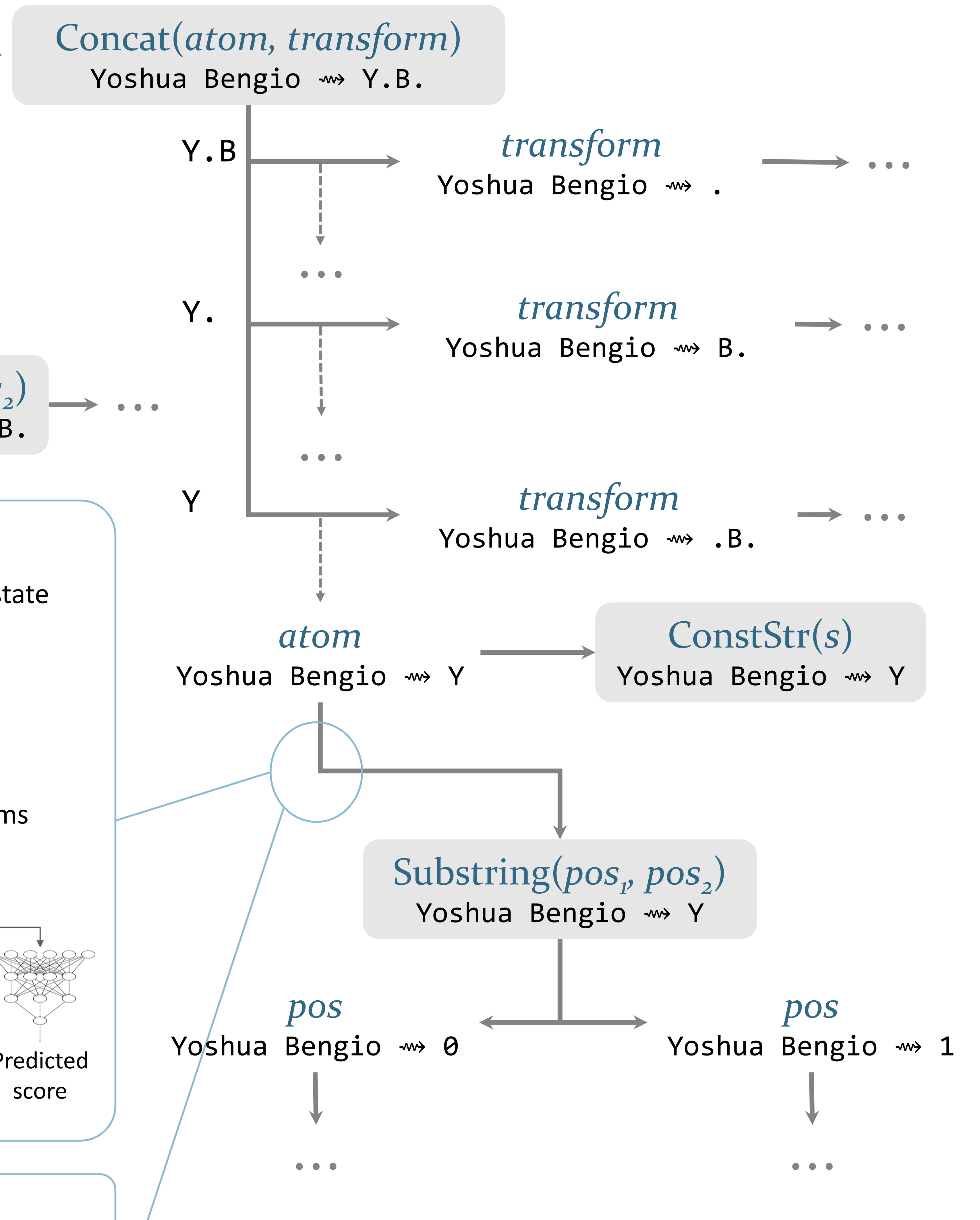
Symbolic: *deductive search*

- Derive the desired program top-down
- Logically reduce the synthesis problem into smaller subproblems
- Ensures that the derived programs satisfy the examples

Neural: *branch prioritization*

- A model to eliminate most unproductive search branches *a priori*
- Incorporates statistical insight about real-life programs
- Improves search performance up to 12 ×





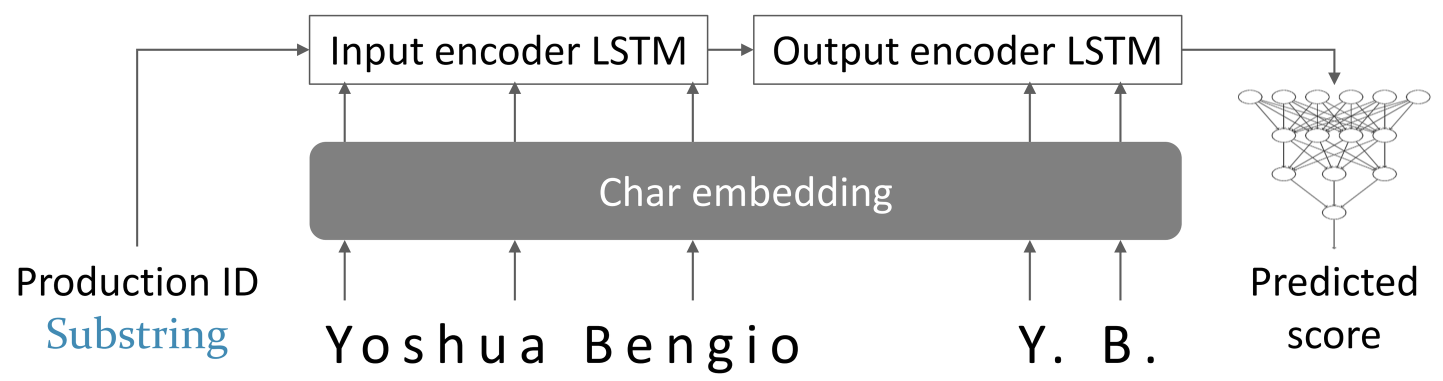
Branch prioritization model

- At every branching node $N := F_1 \mid F_2$, given the search state
- *Predict* the generalization score of the best program P produced from F_1 or F_2:

$$f(\text{branch } \Gamma, \text{spec } \varphi) \approx \max_{P \in \text{Learn}(\Gamma, \varphi)} \text{score}(P, \varphi)$$

Benefits:

- Guides the search process toward higher-quality programs
- Eliminates unproductive search branches



Branch selection controller

Exploiting predictions is a *performance/generalization* trade-off:

- Too few branches ⟹ fast, fragile, likely non-generalizable
- Too many branches ⟹ slow, no statistical insight

Solution: a *controller* based on branch & bound optimization.

Failing scenario (0.5 × speed-up)

Input	Output
41.7114830017,-91.41233825683, 41.60762786865,-91.63739013671	41.7114830017

Training

- Deductive search generates *independent* subproblems
- The search space without neural guidance is exponential
- Thus, 375 real-world tasks ⟹ over 400,000 subproblems:

$$\langle \text{Branch } \Gamma, \text{subproblem spec } \varphi, \text{a posteriori best program score} \rangle$$

- Fully supervised regression training, squared-error loss

Evaluation

Industrial setup: generalize from a *single* input-output example.

Goal: match or exceed PROSE *accuracy* + significantly reduce its *time*.

Baselines:

- PROSE [Polozov & Gulwani 2015], purely symbolic
- RobustFill [Devlin et al. 2017] with 1-3 examples, purely neural
- DeepCoder [Balog et al. 2017] with 1-3 examples, a *shallow* neuro-symbolic hybrid

Method	Accuracy %	Speed-up × PROSE
PROSE	67.12	1.00
DeepCoder, 1 ex.	35.81	**1.82**
DeepCoder, 2 ex.	47.38	1.53
DeepCoder, 3 ex.	62.92	1.42
RobustFill, 1 ex.	24.53	0.25
RobustFill, 2 ex.	39.72	0.27
RobustFill, 3 ex.	56.41	0.30
NGDS	**68.49**	**1.67**

Purely neural or purely symbolic systems suffer in accuracy and/or speed.

Successful scenario (2.7 × speed-up)

Input	Output
457 124th St S, Seattle, WA 98111	Seattle-WA