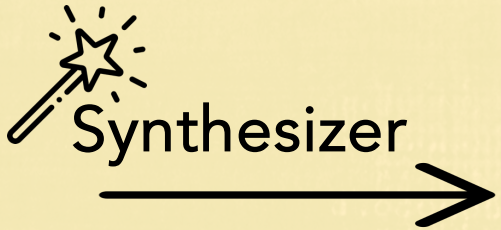
Fast and Reliable Program Synthesis via User Interaction

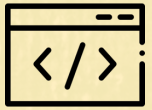
Yanju Chen, Chenglong Wang, Xinyu Wang, Osbert Bastani, Yu Feng

UCSB Microsoft Research UNIVERSITY OF MICHIGAN Penn

Program Synthesis

Specification φ





Program P

Find a program P that satisfies the specification φ.

Natural Languages

Input-Output Examples

Logical Constraints

...

Specifications

How does a synthesizer prune the search space of *incorrect* candidate programs and nail down *promising* ones?

Motivating Example

$e_{in} : [1, 3, 5, 2, 4] \mapsto e_{out} : [5, 4, 3]$

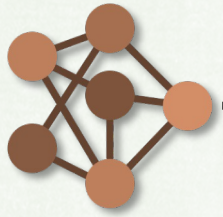
IO Example as Specification

$\hat{e}_{in}$ Size: 5

$\hat{e}_{out}$ Size: 3

Abstract Specification

What would synthesizers usually do?



Sampling

Statistical Policy

Candidate Partial Program

`reverse(map(`e_{in}`, ◯))`

$\hat{e}_{in}$ 5

Abstract Reasoning

5

$\hat{e}_{out}$ 3

Deduction Engine

Inconsistent; pruned

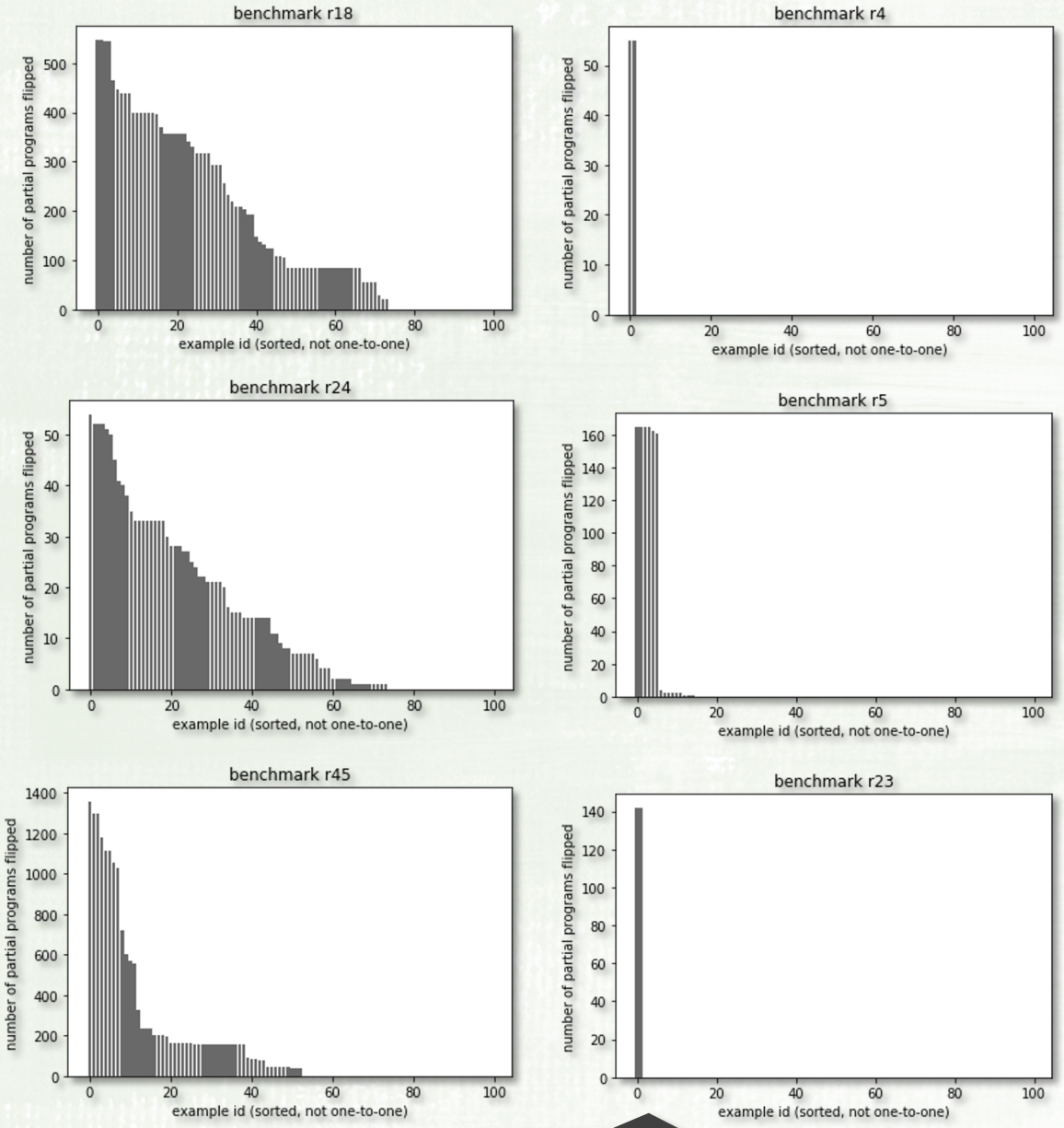
Motivations

Quality of Different IO Examples

It's very difficult for end-users without proper expertise to provide *good* IO examples.

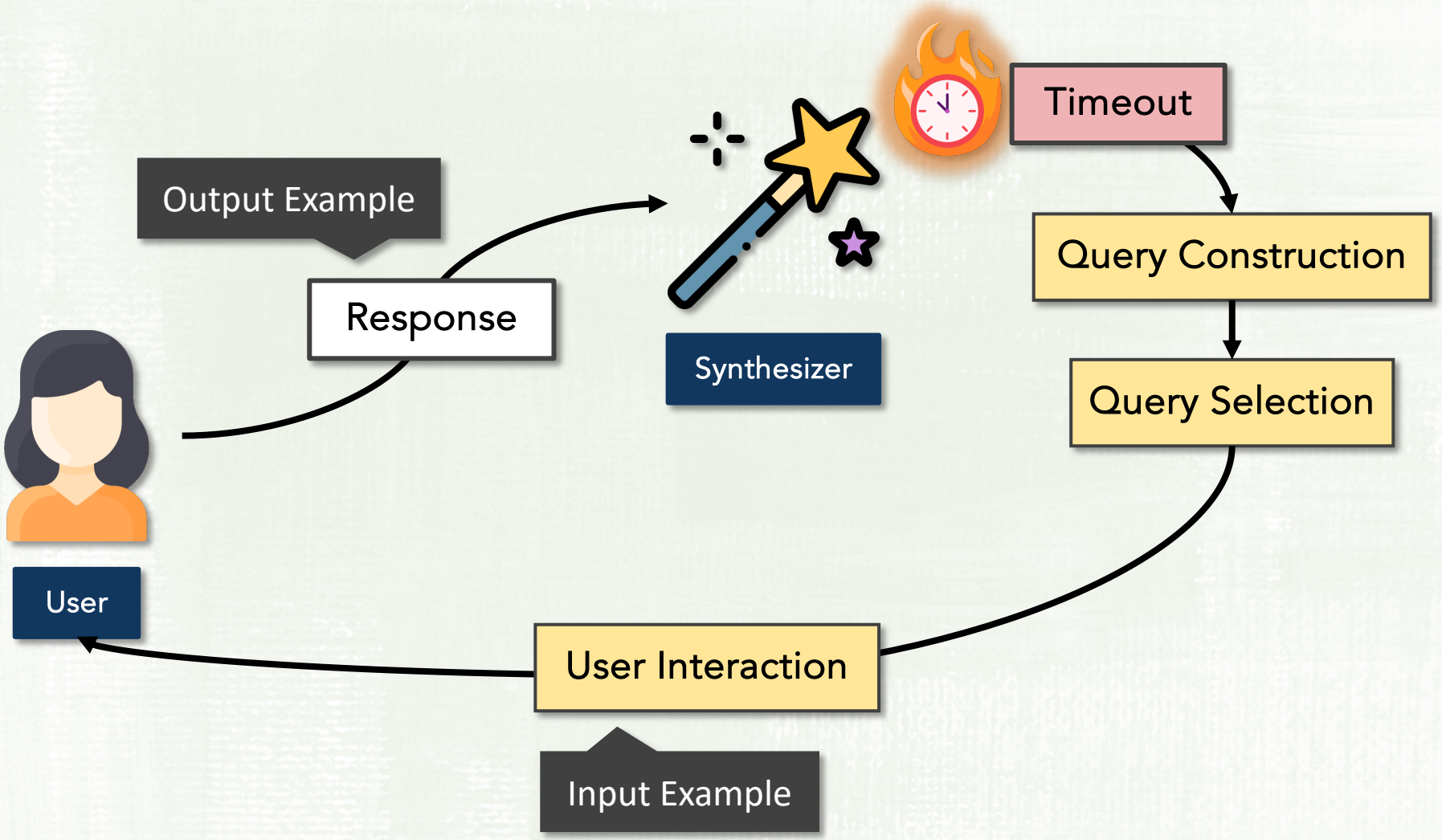
Let the synthesizer guide the user in providing useful IO examples.

via User Interaction

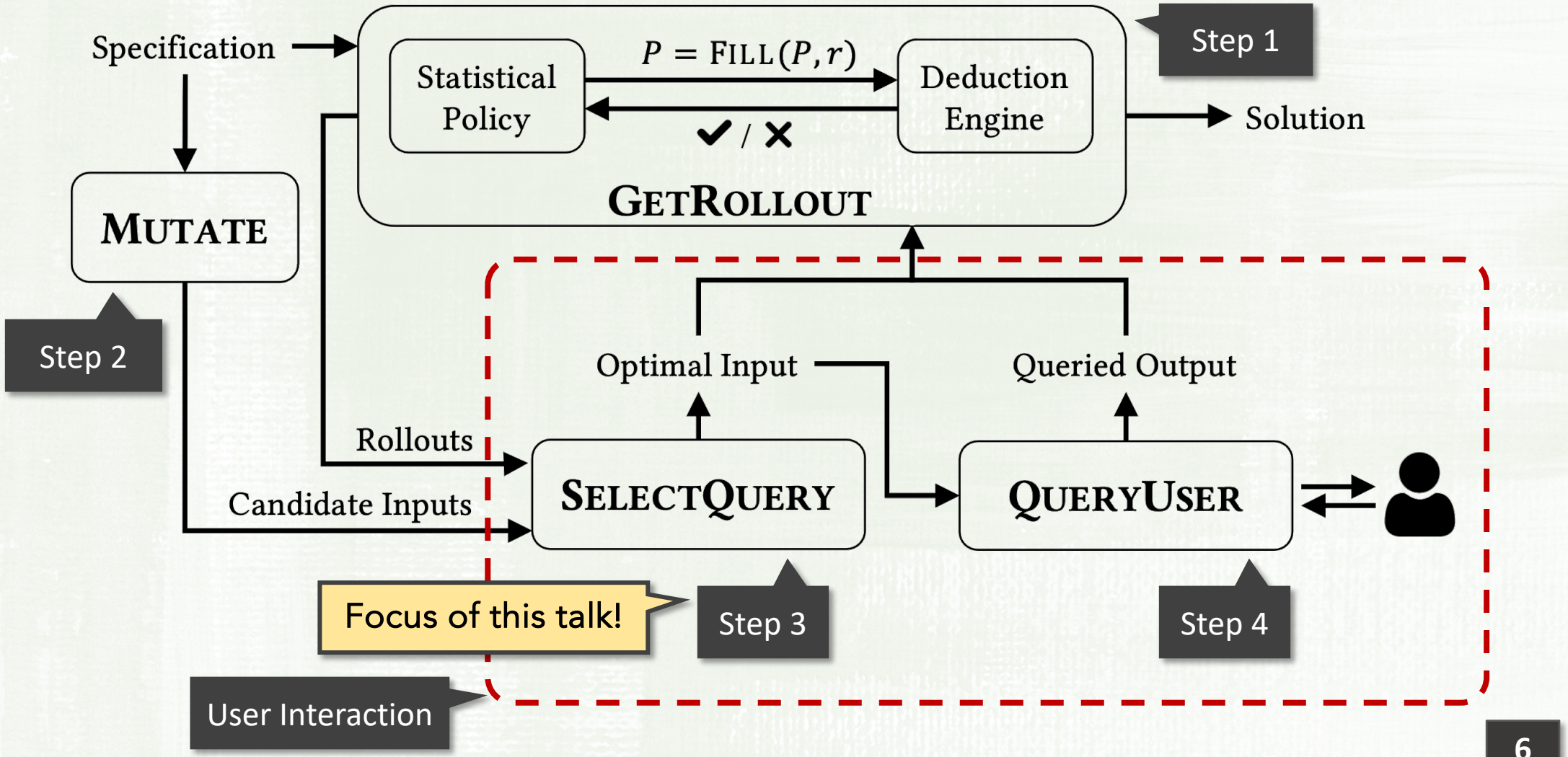


Real-World Majority

User Interaction



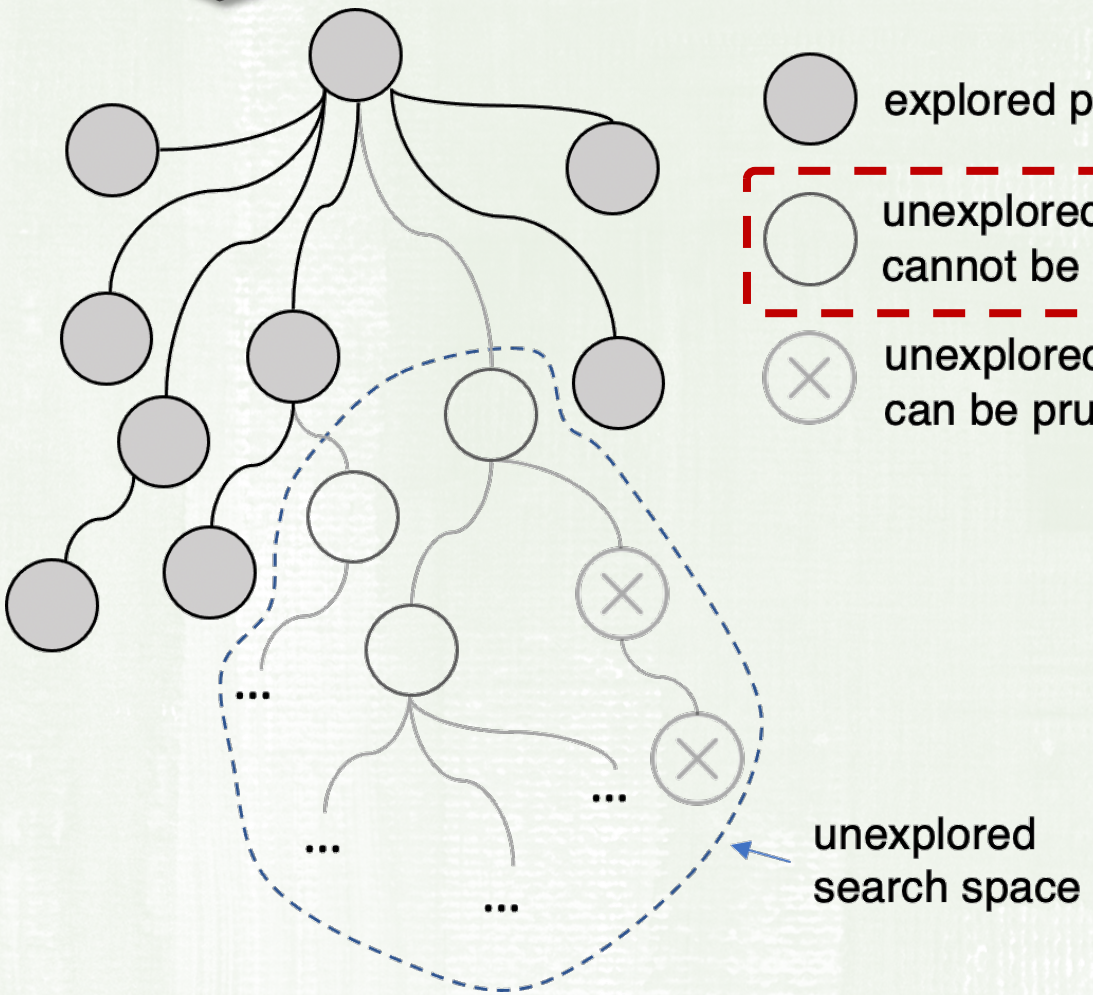
Overview of FAERY



Selection of Best Query

How a structured search is performed:

Input Example



explored programs

Target Search Space

unexplored programs that cannot be pruned by $\langle e_{in}, e_{out} \rangle$

Search space used to evaluate quality of query

unexplored programs that can be pruned by $\langle e_{in}, e_{out} \rangle$

unexplored search space

How?

Given correct user response, pick the query *expected* to prune the largest search space that *can't* be pruned by the *initial* example.

Selection of Best Query (cont'd)

How do we evaluate the quality of a query?

Sampled Target Search Space

Candidate Input

e'_{in}

⟨6,[1,5],1,4⟩

Sampled Programs

P_1: `take(reverse(sort(`e_{in}`)),3)`

P_2: `drop(reverse(sort(`e_{in}`)),2)`

P_3: `take(reverse(map(`e_{in}`,+1)),3)`

Unexplored programs that can't be pruned by initial specification

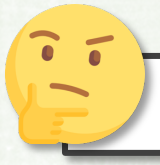
Abstract Reasoning

Abstract Output	P_1	P_2	P_3	Objective Value
⟨3,[1,5],5,[1,5]⟩	✓	✗	✓	2
⟨4,[1,5],5,[1,5]⟩	✗	✓	✗	1
⟨3,[2,6],5,2⟩	✓	✗	✓	2

Aggregated Objective Value

$J^{\mathcal{A}}_{abs} = 2$

"Quality" of the candidate input/query



How?

Given correct user response, pick the query *expected* to prune the largest search space that *can't* be pruned by the *initial* example.

Evaluation Setup

We instantiate and evaluate FAERY on two domains.

Data Wrangling — Adapted DSL and benchmarks from previous works[1,2].

JSON Transformation — Adapted DSL from JQ[3] library; Collected benchmarks from StackOverflow.

We compare FAERY with state-of-the-art tools.

NEO[1] — For data wrangling domain, we directly compare with the tool.

TRINITY[4] — We build a JSON transformation version of the tool.

[1] **Program Synthesis Using Conflict-Driven Learning**. Feng, Y. et al. PLDI'18.
[2] **Component-Based Synthesis of Table Consolidation and Transformation Tasks from Examples**. Feng, Y. et al. PLDI'17.
[3] **Trinity: An Extensible Synthesis Framework for Data Science**. Martins, R. et al. VLDB'19.
[4] **JQ: a lightweight and flexible command-line json processor**. Dolan, S. 2018.

Evaluation Results

Benchmark		MAX
Data Wrangling	#solved	14/15
	avg. time	174s
	avg. speed-up	2.4×
JSON Transformation	#solved	15/15
	avg. time	48s
	avg. speed-up	9.5×

Table: Scalability Improvements

Strategy	Data Wrangling	JSON Transformation
FAERY (MAX)	99s	105s
FAERY (RANDOM)	142s	168s
NEO/TRINITY	170s	174s

Table: Reliability Improvements

FAERY is effective; check paper for more results.

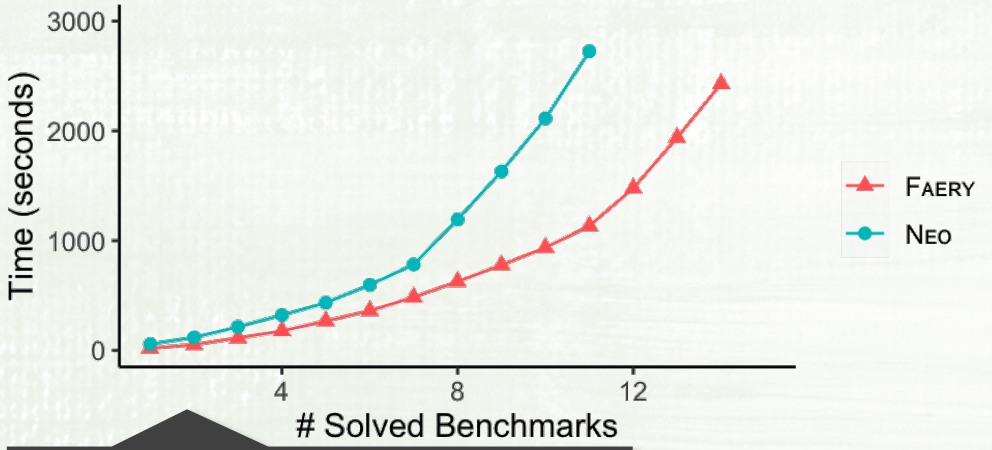


Figure: Comparison with NEO on *data wrangling* domain

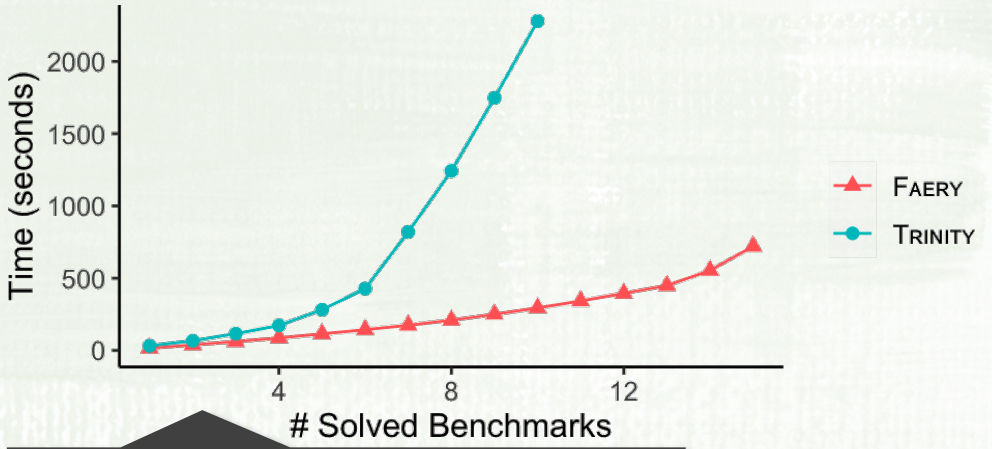


Figure: Comparison with TRINITY on *JSON transformation* domain

Conclusions

Program synthesizer improvement via user interaction

A novel interactive synthesis algorithm

Empirically demonstrated benefits of proposed algorithm

Questions?