Learning to synthesize programs from informative examples

Saujas Vaduguru

Machine learning and code

[1]

[2]

Voyager[3]

Environment Feedback

I cannot make stick because I need: 2 more planks

I cannot make stone_shovel because I need: 2 more stick

GPT-4

```
async function craftStoneShovelWithTable(bot) {
  // If not enough cobblestone, mine cobblestone
  ...
+ // If not enough sticks, check if there are
+ // enough planks in the inventory
  if (sticksCount < 2) {
+   const planksCount = bot.inventory.count(
+     mcData.itemsByName.acacia_planks.id);
+   if (planksCount < 2) {
+     // Collect or craft planks
+     await mineBlock(bot, "acacia_log", 1);
+     await craftItem(bot, "acacia_planks", 1);
+   }
+   // Craft sticks using planks
    await craftItem(bot, "stick", 1);}
  await craftItem(bot, "stone_shovel", 1);
}
```

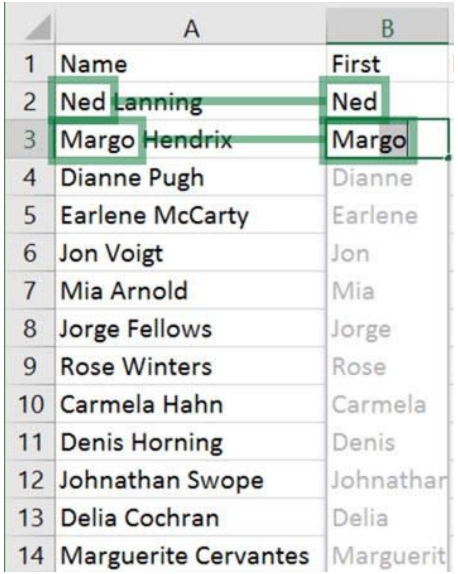
Programming-by-example

Find a `program` that produces behavior consistent with a given set of *examples*

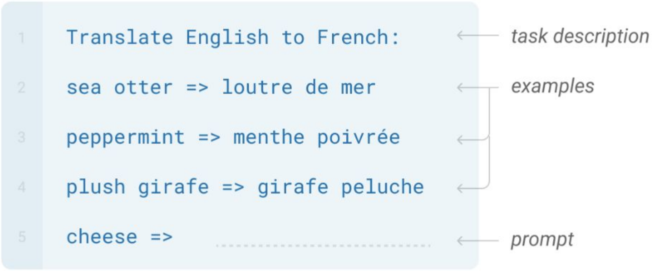
List Processing	Text Editing	Regexes
Sum List	**Abbreviate**	**Phone numbers**
[1 2 3] → 6	Allen Newell → A.N.	(555) 867-5309
[4 6 8 1] → 17	Herb Simon → H.S.	(650) 555-2368
Double	**Drop Last Three**	**Currency**
[1 2 3] → [2 4 6]	shrdlu → shr	$100.25
[4 5 1] → [8 10 2]	shakey → sha	$4.50
Check Evens	**Extract**	**Dates**
[0 2 3] → [T T F]	a b (c) → c	Y1775/0704
[2 9 6] → [T F T]	a (bee) see → see	Y2000/0101

Examples courtesy Ellis *et al.* [4]

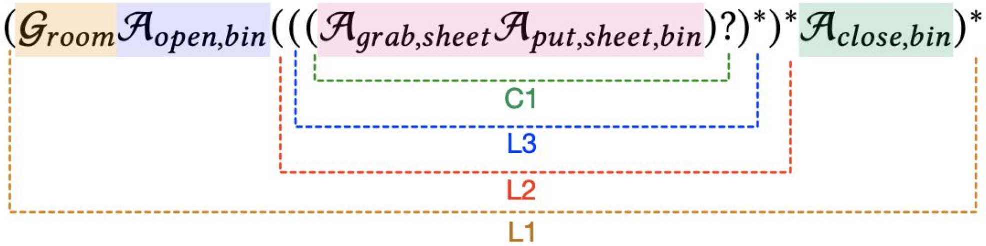
Communicating by example



[5]



[6]



[7]

Ambiguity

Given the examples (*(555) 867-5309*, ✔), (*(650) 555-2368*, ✔), we would like a synthesizer to infer `([0-9]{3}) [0-9]{3}-[0-9]{4}`

But what about

- `([0-9]+) [0-9]+-[0-9]+`?
- `([0-9]5[0-9]) [0-9]{3}-[0-9]{4}`?
- `(.{3}) .{3}-.{4}`?

Reasoning about ambiguity in communication games

Thinking about programming-by-example as a *communication game* can help us reason about ambiguity

- Cooperative game
- *Speaker* chooses examples to communicate a program
- *Listener* infers a program given examples
- Both players win when listener infers the intended program
- Both players share knowledge of program semantics

A Bayesian perspective

$$P_{\text{listener}}(\texttt{program} \mid \textit{examples}) \propto P_{\text{speaker}}(\textit{examples} \mid \texttt{program}) P(\texttt{program})$$

A straightforward approach to program synthesis

prior/ranking/score

$$P_{\text{listener}}(\texttt{program} \mid \textit{examples}) \propto \boxed{\mathbf{1}[\texttt{program} \vdash \textit{examples}]\, P(\textit{examples})}\, \boxed{P(\texttt{program})}$$

truthful/correct speaker

A pragmatic approach to program synthesis

$$P_{\text{pragmatic-listener}}(\texttt{program}|\textit{examples}) \propto P_{\text{speaker}}(\textit{examples}|\texttt{program})P(\texttt{program})$$

$$P_{\text{speaker}}(\textit{examples}|\texttt{program}) \propto P_{\text{listener}}(\texttt{program}|\textit{examples})P(\textit{examples})$$

Rational Speech Acts!

Rational speech acts for programs

$\mathbf{1}[\mathrm{p} \vdash e]$

	a+b	a{2,}b
$(aab, \checkmark)$	1	1
$(ab, \checkmark)$	1	0

$P_{\text{listener}}(\mathrm{p}|e)$

	a+b	a{2,}b
$(aab, \checkmark)$	0.5	0.5
$(ab, \checkmark)$	1	0

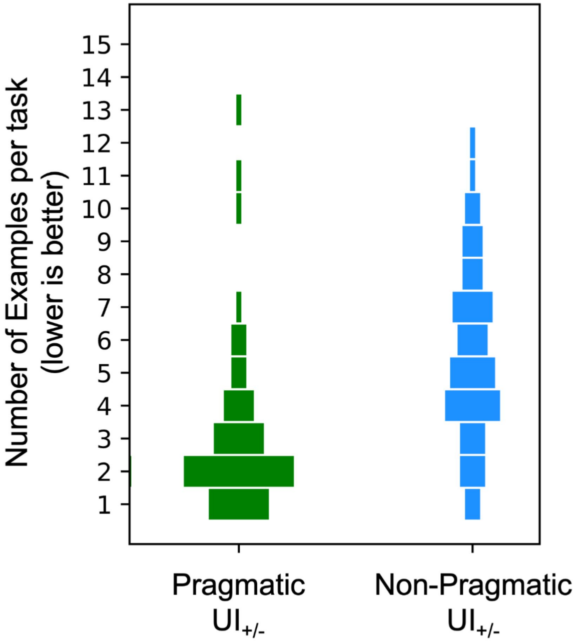
$P_{\text{speaker}}(e|\mathrm{p})$

	a+b	a{2,}b
$(aab, \checkmark)$	0.33	1
$(ab, \checkmark)$	0.67	0

$P_{\text{pragmatic-listener}}(\mathrm{p}|e)$

	a+b	a{2,}b
$(aab, \checkmark)$	0.25	0.75
$(ab, \checkmark)$	1	0

Pragmatic inference makes for a better synthesizer



[8]

Challenges to scaling up

$$P_{\text{speaker}}(e|\mathrm{p};\mathbf{P},\mathbf{E}) = \frac{P_{\text{listener}}(\mathrm{p}|e)P(e)}{\sum_{e' \in \mathbf{E}} P_{\text{listener}}(\mathrm{p}|e')P(e')}$$

$$P_{\text{pragmatic-listener}}(\mathrm{p}|e;\mathbf{P},\mathbf{E}) = \frac{P_{\text{speaker}}(e|\mathrm{p})P(\mathrm{p})}{\sum_{\mathrm{p}' \in \mathbf{P}} P_{\text{speaker}}(e|\mathrm{p}')P(\mathrm{p}')}$$

Scaling up pragmatic inference with rankings

Amortizing Pragmatic Program Synthesis with Rankings
Yewen Pu, Saujas Vaduguru, Priyan Vaithilingam, Elena Glassman, Daniel Fried
International Conference on Machine Learning (ICML), 2024
https://arxiv.org/abs/2407.02499

Rankings over programs

$$P_{\text{pragmatic-listener}}(\mathsf{p}_1|e) > P_{\text{pragmatic-listener}}(\mathsf{p}_2|e) > \cdots > P_{\text{pragmatic-listener}}(\mathsf{p}_n|e)$$

$$\Rightarrow \mathsf{p}_1 \succ \mathsf{p}_2 \succ \cdots \succ \mathsf{p}_n$$

$$\mathsf{p}_1^* \succ \mathsf{p}_2^* \succ \cdots \succ \mathsf{p}_n^* \textbf{?}$$

Amortizing pragmatics with rankings

$$\texttt{target} \sim P(\texttt{program}) \qquad \textit{examples} = \text{argmax}_e\, P_{\text{speaker}}(e|\texttt{target})$$

$$P_{\text{pragmatic-listener}}(\mathsf{p}_1|\textit{examples}) > P_{\text{pragmatic-listener}}(\mathsf{p}_2|\textit{examples}) > \cdots > P_{\text{pragmatic-listener}}(\mathsf{p}_n|\textit{examples})$$
$$\Rightarrow \sigma = \mathsf{p}_1 \succ \mathsf{p}_2 \succ \cdots \succ \mathsf{p}_n$$

$$\{(\texttt{target}, \textit{examples}, \sigma)\}$$

Inferring a global ranking

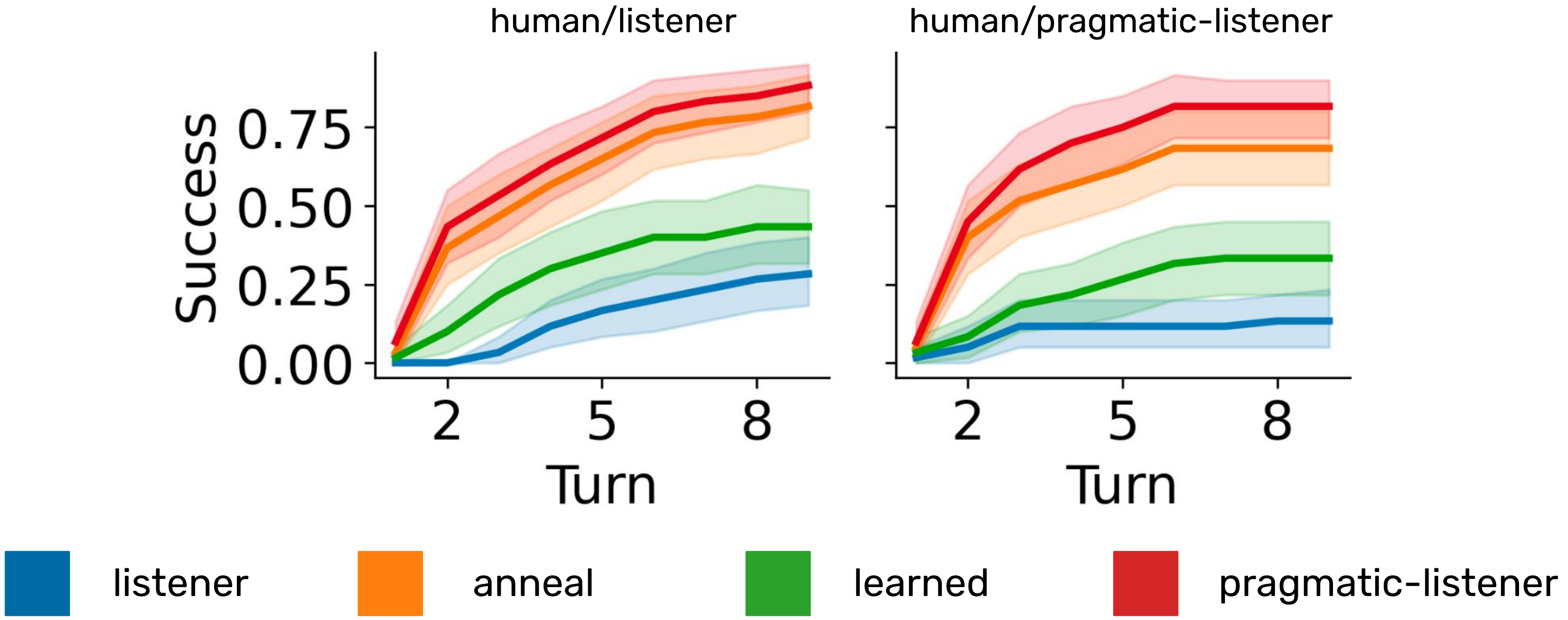
Annealing

$$\begin{gathered}\sigma^* = \langle \mathsf{p}_1^*, \mathsf{p}_2^*, \cdots, \mathsf{p}_n^* \rangle \\ \mathsf{p}_2 \succ \mathsf{p}_1 \in \sigma \sim \{(\texttt{target}, \textit{examples}, \sigma)\} \\ \Rightarrow \sigma^*[\mathsf{p}_1] \leftrightharpoons \sigma^*[\mathsf{p}_2]\end{gathered}$$

Learned score function

$$\mathrm{argmin}_\theta\, \mathbb{E}_{\{(\texttt{target}, \textit{examples}, \sigma)\}} \left[- \log \left(\mathrm{sig}(s_\theta(\mathsf{p}_i) - s_\theta(\mathsf{p}_j))\right)\right]$$

Rankings improve synthesis for binary regexes



Training program synthesizers on pragmatic examples

Generating Pragmatic Examples to Train Neural Program Synthesizers

Saujas Vaduguru, Daniel Fried, Yewen Pu

International Conference on Learning Representations (ICLR), 2024

https://arxiv.org/abs/2311.05740

Machine learning for programming-by-example

$$P_{\text{listener}}(\texttt{program}|\textit{examples}) \propto \mathbf{1}[\texttt{program} \vdash \textit{examples}]P_\theta(\texttt{program}|\textit{examples})$$

$$\begin{aligned} P(\textit{example}|\texttt{program}) &= P(\textit{input}, \textit{output}|\texttt{program}) \\ &= P(\textit{output}|\textit{input}, \texttt{program})P(\textit{input}|\texttt{program}) \\ &= \mathbf{1}[\texttt{program}(\textit{input}) = \textit{output}]P(\textit{input}) \end{aligned}$$

Literal training recipe: [9, 10]

$$\textit{input} \sim P(\textit{input}), \texttt{program} \sim P(\texttt{program}) \rightarrow (\{(\textit{input}, \textit{output})\}, \texttt{program}) \rightarrow \text{fit } \theta$$

PraX: Generating pragmatic examples

$$\texttt{target} \sim P(\texttt{program})$$

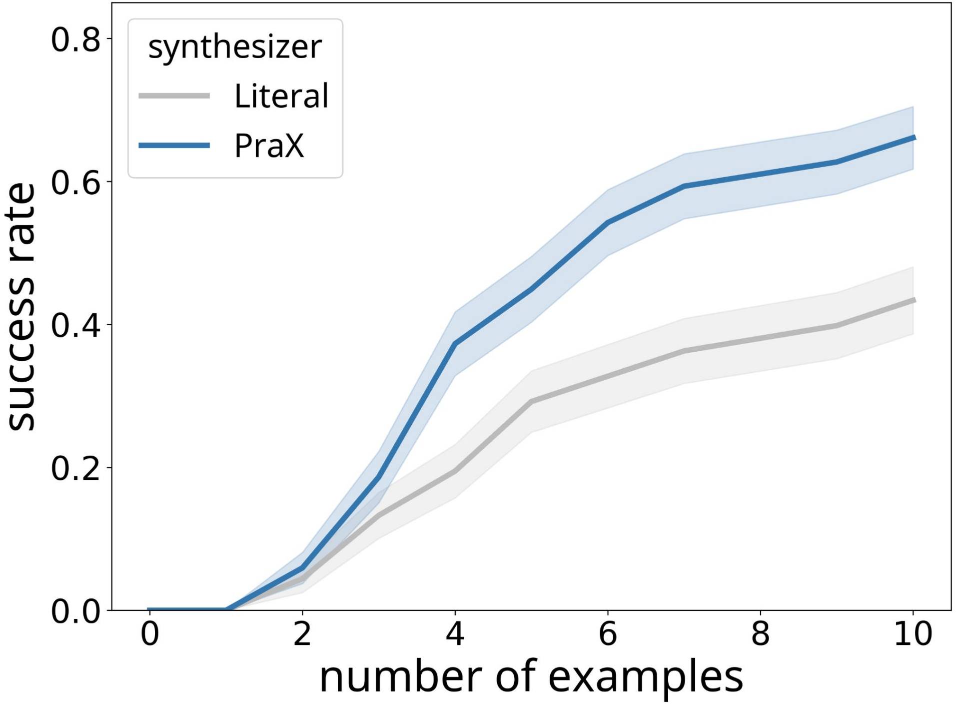
$$\text{EXAMPLES} \sim P_\phi(\textit{examples} | \texttt{target}) \qquad \texttt{GUESSES} \sim P_\theta(\texttt{program} | \textit{examples})$$

$$\textit{examples}^* = \text{argmax}_{e \in \text{EXAMPLES}} P_{\text{speaker}}(e | \texttt{target}; \texttt{GUESSES} \cup \{\texttt{target}\}, \text{EXAMPLES})$$

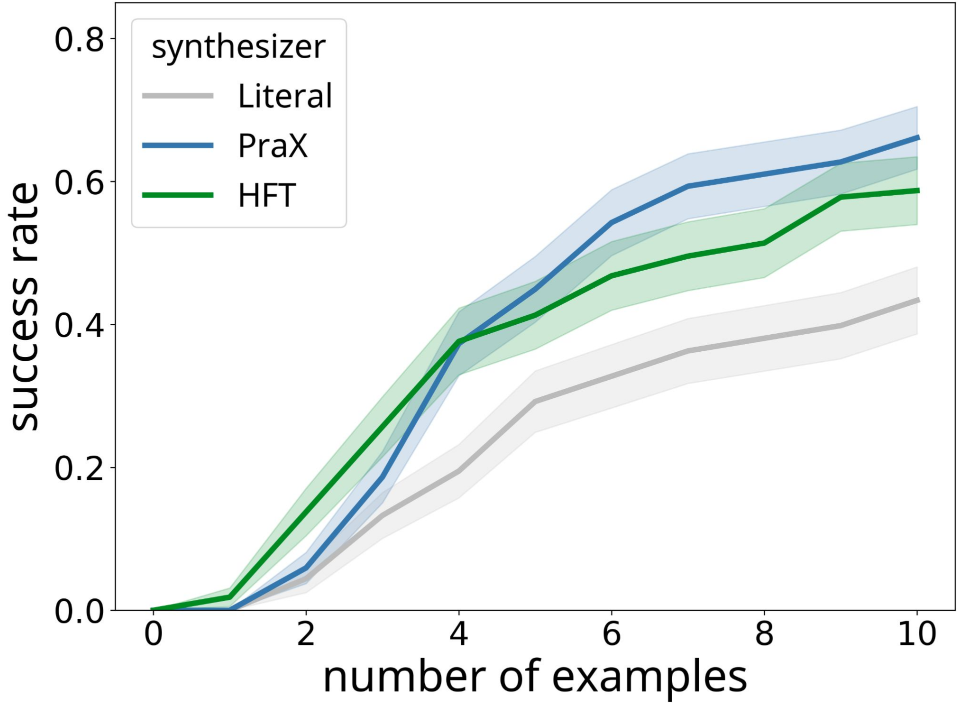
$$\{(\textit{examples}^*, \texttt{target})\}$$

$$\text{fit } \phi, \theta$$

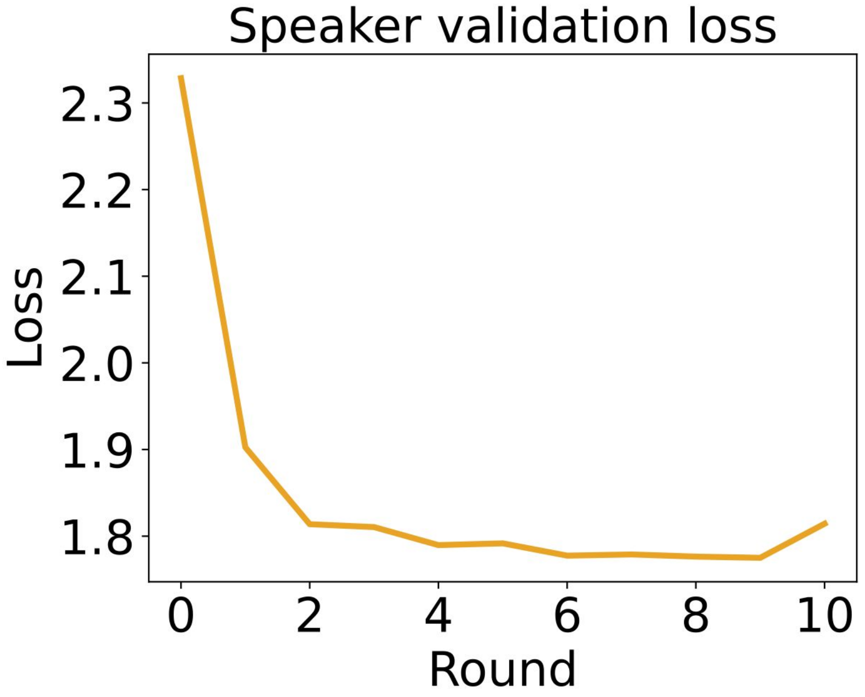
PraX outperforms literal training



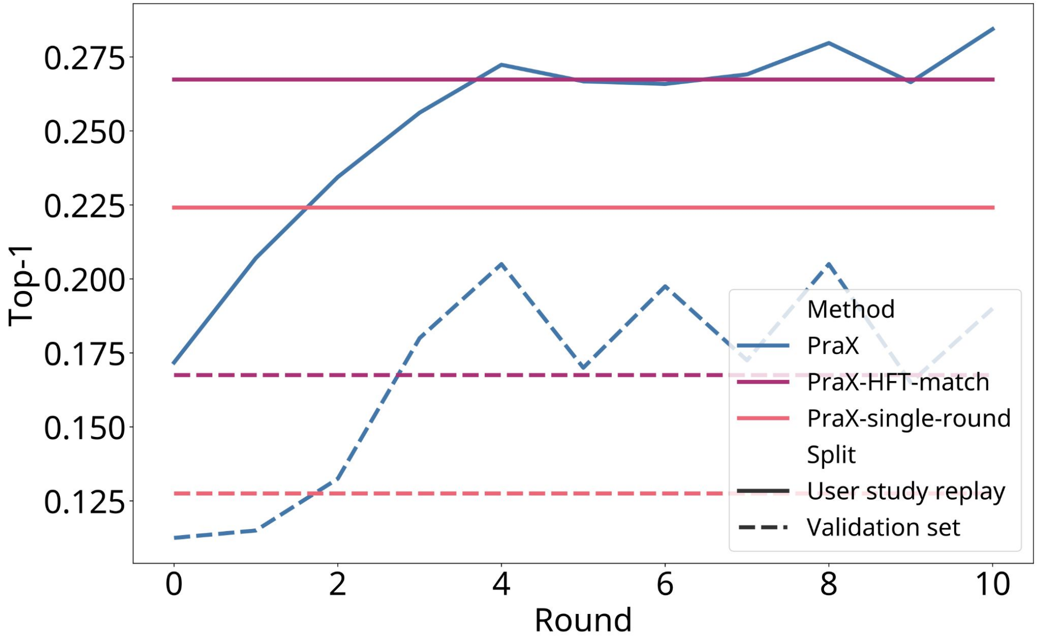
PraX outperforms fine-tuning on human-provided examples



Speaker model learns more human-like behavior



Speaker learns to produce higher-quality examples over rounds of training



Learning from multi-agent interaction for pragmatics

- PraX shows how we can simulate interactions between agents to make them better at communicating with humans
- Learning in prover-verifier games has been shown to make proofs more *legible*[11]

Takeaways

- Paying attention to the kind of reasoning that generates inputs is effective
- Cognitive science can guide the way we synthesize data to train models rather than directly inspire model design choices

Reach out!

saujasv.github.io

saujasv@cmu.edu

References

[1] Zhuo, T. Y., Vu, M. C., Chim, J., Hu, H., Yu, W., Widyasari, R., ... & Von Werra, L. (2024). Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. arXiv preprint arXiv:2406.15877.

[2] Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., & Narasimhan, K. (2023). Swe-bench: Can language models resolve real-world github issues?. arXiv preprint arXiv:2310.06770.

[3] Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., ... & Anandkumar, A. (2023). Voyager: An open-ended embodied agent with large language models. arXiv preprint arXiv:2305.16291.

[4] Ellis, K., Wong, C., Nye, M., Sable-Meyer, M., Cary, L., Morales, L., … Tenenbaum, J. B. (2020). DreamCoder: Growing generalizable, interpretable knowledge with wake-sleep Bayesian program learning. arXiv [Cs.AI]. Retrieved from http://arxiv.org/abs/2006.08381

[5] Gulwani, S. (2011). Automating string processing in spreadsheets using input-output examples. ACM Sigplan Notices, 46(1), 317-330.

[6] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., … Amodei, D. (2020). Language Models are Few-Shot Learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, & H. Lin (Eds.), Advances in Neural Information Processing Systems (Vol. 33, pp. 1877–1901).

[7] Patton, N., Rahmani, K., Missula, M., Biswas, J., & Dillig, I. (2024). Programming-by-demonstration for long-horizon robot tasks. Proceedings of the ACM on Programming Languages, 8(POPL), 512-545.

[8] Vaithilingam, P., Pu, Y., & Glassman, E. L. (2023). The Usability of Pragmatic Communication in Regular Expression Synthesis. arXiv preprint arXiv:2308.06656.

[9] Devlin, J., Uesato, J., Bhupatiraju, S., Singh, R., Mohamed, A. R., & Kohli, P. (2017, July). Robustfill: Neural program learning under noisy i/o. In International conference on machine learning (pp. 990-998). PMLR.

[10] Balog, M., Gaunt, A. L., Brockschmidt, M., Nowozin, S., & Tarlow, D. (2016). Deepcoder: Learning to write programs. arXiv preprint arXiv:1611.01989.

[11] Kirchner, J. H., Chen, Y., Edwards, H., Leike, J., McAleese, N., & Burda, Y. (2024). Prover-verifier games improve legibility of llm outputs. arXiv preprint arXiv:2407.13692.