On Formulating and Evaluating Language Agents

Shunyu Yao
Use LLMs to interact with the world
<table>
<thead>
<tr>
<th>A lot of Terms</th>
<th>A lot of Papers</th>
<th>A lot of Products</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Language agent</td>
<td>● SayCan</td>
<td>● ChatGPT plugins</td>
</tr>
<tr>
<td>● LLM-empowered agents</td>
<td>● ReAct</td>
<td>● Windows copilot</td>
</tr>
<tr>
<td>● LLM powered autonomous agents</td>
<td>● Toolformer</td>
<td>● Perplexity search</td>
</tr>
<tr>
<td>● Language enabled agents</td>
<td>● Generative Agents</td>
<td>● LangChain</td>
</tr>
<tr>
<td>● LLM based agents</td>
<td>● Tree of Thoughts</td>
<td>● Adept ACT-1</td>
</tr>
<tr>
<td>● ......</td>
<td>● ......</td>
<td>● ......</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A lack of Theories</th>
<th>A lack of Benchmarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>● What defines “language agent”?</td>
<td></td>
</tr>
<tr>
<td>● How to unify existing efforts?</td>
<td></td>
</tr>
<tr>
<td>● What is lacking?</td>
<td>● NLP tasks? (too easy to solve?)</td>
</tr>
<tr>
<td></td>
<td>● Robotics tasks? (too hard to set up?)</td>
</tr>
<tr>
<td></td>
<td>● Evaluation? (too noisy and subjective?)</td>
</tr>
</tbody>
</table>
Part 1. Formulation
ReAct (Yao et al., 2022)

Thought 1: I need to search Apple Remote and find the program it was originally designed to interact with.
Act 1: Search[Apple Remote]
Obs 1: The Apple Remote is a remote control introduced in October 2005 by Apple ... originally designed to control the Front Row media center program ...

Thought 2: Apple Remote was originally designed to control the Front Row media center program. I need to search Front Row next and find what other device can control it.
Act 2: Search[Front Row]
Obs 2: Could not find [Front Row]. Similar: ['Front Row Seat to Earth', 'Front Row Motorsports', 'Front Row (software)', ...]

Thought 3: Front Row is not found. I need to search Front Row (software).
Act 3: Search[Front Row (software)]
Obs 3: Front Row is a discontinued media center software ...

Thought 4: Front Row (software) is controlled by an Apple Remote or the keyboard function keys. So the answer is keyboard function keys.
Act 4: Finish[Keyboard function keys]
Voyager (Wang et al., 2023)
Generative Agents (Park et al., 2023)
How to **make sense** of Language Agents?

- Control flow
- LLM
- Prompt engineering
- memory
- planning
- reasoning
- Decision making
- self-reflection
- robotics
- web
- multi-agent
- environment
- acting
- coding
- grounding
- retrieval
- feedback
Circuits

How to make sense of Language Agents?
Von Neumann architecture makes sense & guides building of circuits.
How to make sense of Language Agents?

A system architecture, with LLM as a component
(computer architecture, with CPU as a component)

Cognitive Architectures for Language Agents

Theodore Sumers*  Shunyu Yao*  Karthik Narasimhan  Thomas L. Griffiths
Princeton University
{sumers, shunyuy, karthikn, tomg}@princeton.edu

- **History**: What’s cognitive architecture?
- **Analog**: Why it’s related to language agents?
- **Framework**: How to formulate language agents via CoALA?
- **Insights**: Future directions through CoALA?
1900-1950: Production System Theorized

- **Symbol manipulation** formalizes math, logic, and computation
- **Production system** formalizes symbol manipulation:
  - a set of precondition $\rightarrow$ action rules $X \ Y \ Z \rightarrow X \ W \ Z$
  - Also used to explain language & cognition
1950-1980: Production System Implemented

- Symbol manipulation → **Physical symbol system**
  
  $\text{(temperature} > 70^\circ) \land (\text{temperature} < 72^\circ) \rightarrow \text{stop}$
  
  $\text{temperature} < 32^\circ \rightarrow \text{call for repairs; turn on electric heater}$
  
  $\text{(temperature} < 70^\circ) \land (\text{furnace off}) \rightarrow \text{turn on furnace}$
  
  $\text{(temperature} > 72^\circ) \land (\text{furnace on}) \rightarrow \text{turn off furnace}$

- **Usage towards real-world applications**
  - Interact with the world → IO devices
  - Many possible actions → Priorities over fired rules
  - Complicated information → Memory mechanisms
  - ...

Production System -> **Cognitive Architectures**

**Issue:** world is...
- **Complex:** too many rules
- **Stochastic:** rules can be fragile
2015 - 2022

**Issue:** world is...
- **Complex:** too many rules
- **Stochastic:** rules can be fragile

LMs are like large, implicit production systems
Language agents can be guided by Cognitive Architectures!
The CoALA framework

1. Memory
2. Action
3. Decision
1. Memory

**Long-term memory**
1. **Episodic** (experience)
2. **Semantic** (knowledge)
3. **Procedural** (LLM, code)

**Short-term working memory**
- Information for the current “decision cycle”
2. Action Space

- A language agent is defined with an action space
  - External actions interact with external environments (**grounding**)
  - Internal actions interact with internal memories
    - **Reasoning**: read & write working memory
    - **Retrieval**: read long-term memory
    - **Learning**: write long-term memory
3. Decision Making

- A language agent chooses actions via **decision (making)** procedures
  - Split taken actions into decision cycles
  - In each cycle, plan then execute a learning/grounding action
    - **Planning**: use reasoning/retrieval to propose/evaluate actions
    - **Execution**: apply the learning/grounding action
That’s it, basically.
Make Sense of (Existing) Language Agents
### Make Sense of (Existing) Language Agents

<table>
<thead>
<tr>
<th>Agent</th>
<th>Long-term Memory</th>
<th>External Grounding</th>
<th>Internal Actions</th>
<th>Decision Making</th>
</tr>
</thead>
<tbody>
<tr>
<td>SayCan (Ahn et al., 2022)</td>
<td>-</td>
<td>physical</td>
<td>-</td>
<td>evaluate</td>
</tr>
<tr>
<td>ReAct (Yao et al., 2022b)</td>
<td>-</td>
<td>digital</td>
<td>reason</td>
<td>propose</td>
</tr>
<tr>
<td>Voyager (Wang et al., 2023a)</td>
<td>procedural</td>
<td>digital</td>
<td>reason/retrieve/learn</td>
<td>propose</td>
</tr>
<tr>
<td>Generative Agents (Park et al., 2023)</td>
<td>episodic/semantic</td>
<td>digital/agent</td>
<td>reason/retrieve/learn</td>
<td>propose</td>
</tr>
<tr>
<td>Tree of Thoughts (Yao et al., 2023)</td>
<td>-</td>
<td>digital (boxed)</td>
<td>reason</td>
<td>propose, evaluate, select</td>
</tr>
</tbody>
</table>
Learn = Write long-term memory

- Unify various things
Guide Building of (Future) Language Agents

Learn = Write long-term memory

- Unify various things
- Reveal lots of gaps

Updating episodic memory with experience. It is common practice for RL agents to store episodic trajectories to update a parametric policy (Hendryck et al., 2016; Pritzel et al., 2017) or establish a non-parametric policy (Ecoffet et al., 2019; Thiel et al., 2022). For language agents, added experiences in episodic memory may be retrieved later as examples and bases for reasoning or decision making (Winston et al., 2014; Rubin et al., 2021; Park et al., 2023).

Updating semantic memory with knowledge. Recent work (Shinn et al., 2023; Park et al., 2023) has applied LLMs to reason about raw experiences and store the resulting inferences in semantic memory. For example, Reflection (Shinn et al., 2023) uses an LLM to reflect on failed episodes and store the results (e.g., “there is no dishwasher in kitchen”) as semantic knowledge to be attached to LLM context for solving later episodes. Finally, work in robotics (Chen et al., 2023) uses vision-language models to build a semantic map of the environment, which can later be queried to execute instructions.

Updating LLM parameters (procedural memory). The LLM weights represent implicit procedural knowledge. These can be adjusted to an agent’s domain by fine-tuning during the agent’s lifetime. Such fine-tuning can be accomplished via supervised or imitation learning (Hussein et al., 2017), reinforcement learning (RL) from environment feedback (Sutton and Barto, 2018), human feedback (RLHF) (Christiano et al., 2017; Ouyang et al., 2022; Nakano et al., 2022), or AI feedback (Hai et al., 2022). For example, XTX (Thiel et al., 2022) periodically fine-tune a small language model on high-scoring trajectories stored in episodic memory, which serves as a robust “exploitation” policy to reach exploration frontiers in the face of stochasticity. Recent work (Huang et al., 2022c; Zellmann et al., 2022) has also shown the potential of fine-tuned small language models distilling them surpassing larger ones. Fine-tuning the agent’s LLM is a costly form of learning, thus, present studies specify learning schedules. However, as training becomes more efficient – or if agents utilize smaller subtask-specific LLMs – it may be possible to allow language agents to autonomously determine when and how to fine-tune their LLMs.

Updating agent code (procedural memory). CoALA allows agents to update their source code, thus modifying the implementation of various procedures. These can be broken down as follows:

- Updating reasoning (e.g., prompt templates; Guo et al., 2020; Zhou et al., 2023). For example, APE (Zhou et al., 2023) infers prompt instructions from input-output examples, then uses these instructions as part of the LLM prompt to assist task solving. Such a prompt update can be seen as a form of learning to reason.

- Updating grounding (e.g., code-based skills; Liang et al., 2023c; Ellis et al., 2021; Wang et al., 2023c). For example, Voyager (Wang et al., 2023a) maintains a curriculum library. Notably, current methods are limited to creating new code skills to interact with external environments.

- Updating retrieval. To our knowledge, these learning options are not studied in recent language agents. Retrieval is usually considered a basic action designed with some fixed implementation (e.g., BM25 or dense retrieval), but research in query/document expansion (Nogueira et al., 2019; Wang et al., 2023a; Tang et al., 2023b) or retrieval distillation (Laasard et al., 2021) may be helpful for language agents to learn better retrieval procedures.

- Updating learning or decision-making. Finally, it is theoretically possible for CoALA agents to learn new procedures for learning or decision making, thus providing significant adaptability. In general, however, updates to these procedures are risky both for the agent’s functionality and alignment. At present, we are not aware of any language agents that implement this form of learning; we discuss such possibilities more in Section 6.

While RL agents usually fix one way of learning (e.g., Q-learning, PPO, or A3C) and learn by updating model parameters, language agents can select from a diversity of learning procedures. This allows them to learn rapidly by storing task-relevant language (cheaper and quicker than parameter updates), and leverage multiple forms of learning to compound their self-improvement (e.g., Generative Agents discussed in Section 8).

Finally, while our discussion has mostly focused on adding to memory, modifying and deleting (a case of “unlearning”) are understudied in recent language agents. We address these areas more in Section 6.
Guide Building of (Future) Language Agents

1. Prompt engineering
2. Retrieval corpora
3. In-context learning, finetuning
4. External actions, tools

1. Working memory & reasoning
2. Long-term memory
3. Learning
4. Action space

1. Systematic prompting & parsing (e.g. OpenAI function calls, Guidance, ...)
2. Writing to memory, beyond just retrieval (e.g. Reflexion, Generative Agents, Voyager...)
3. Various learning understudied (meta-prompt, update code, unlearn, mixed learning)
4. Synergizing effects & safety analysis via action space
CoALA:
● Agent = LLM + Memory + Code
● Memory = episodic / semantic / procedural (stores LLM+code)
● Code = decision / action procedures
● Action = grounding / reasoning / learning / retrieval

● New techniques can fix old issues
● Old insights can guide new development

Read the paper for much more information!
Part 2. Evaluation
Evaluation of Language Models

0. easy to solve, easy to evaluate  (Most existing NLP tasks are becoming..)
1. easy to solve, hard to evaluate  (Open-ended Text Generation)
2. **hard to solve, easy to evaluate**  (Games; Coding?)
3. hard to solve, hard to evaluate  (Most recent agent tasks?)

Easy to evaluate: cheap, fast, consistent, (not too subjective)
Evaluation: how to be cheap, fast, yet high-quality?

Human Evaluation

😊 High-quality (esp. if you’re OpenAI): RLHF

😢 Not Scalable (if you’re not OpenAI): expensive and slow to collect data

LM Evaluation

😢 Not High-quality (yet): hallucinations

😃 Scalable (somewhat): unlimited, but with costs

“Rule-based” Evaluation

😊 High-quality: if we leverage domain priors!

😢 Scalable: unlimited, free, fast
Evaluation of Language Models

0. easy to solve, easy to evaluate
   (Most existing NLP tasks are becoming..)
1. easy to solve, hard to evaluate
   (Open-ended Text Generation)
2. hard to solve, easy to evaluate
   (Games; Coding?)
3. hard to solve, hard to evaluate
   (Most recent agent tasks?)

Easy to evaluate: cheap, fast, consistent, (not too subjective)

Evaluation of Language Agents

1. **Collie**: make text generation hard to solve and easy to evaluate!
2. **InterCode**: make coding interactive
3. **WebShop**: make web tasks easy to evaluate
Environment: how to be cheap, fast, yet useful?

**Physical World / Humans**
- **Practical**: robots / chatbots
- **Not Scalable**: expensive and slow to collect data

**Digital Simulations / Games**
- **Not Practical**: sim-to-real is hard
- **Scalable**: free, unlimited interactions

**Digital Applications (Internet, code, software, …)**
- **Practical**: important tasks to automate
- **Scalable**: huge scale, rich complexity, free and fast
### Overview

<table>
<thead>
<tr>
<th></th>
<th>1. <strong>WebShop</strong></th>
<th>2. <strong>InterCode</strong></th>
<th>3. <strong>Collie</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Task</strong></td>
<td>“Find a t-shirt that…”</td>
<td>“Remove the file that…”</td>
<td>“Write a paragraph that…”</td>
</tr>
<tr>
<td><strong>Action</strong></td>
<td>Web actions</td>
<td>Code as action</td>
<td>Text as action</td>
</tr>
<tr>
<td><strong>Observation feedback</strong></td>
<td>Webpage</td>
<td>Code execution result</td>
<td>Synthetic constraint satisfaction feedback</td>
</tr>
<tr>
<td><strong>Reward feedback</strong></td>
<td>Attribute rule-based</td>
<td>Unit test or file diff</td>
<td>Grammar rule-based</td>
</tr>
</tbody>
</table>
COLLIE: Systematic Construction of Constrained Text Generation Tasks

Shunyu Yao∗  Howard Chen∗  Austin W. Hanjie∗  Runzhe Yang∗  Karthik Narasimhan
Department of Computer Science, Princeton University
{shunyuy, hc22, hjwang, runzhey, karthikn}@princeton.edu
Constrained Text Generation

- A traditional and important NLP (seq2seq) task: constraints $\rightarrow$ text
- **Prior benchmarks**: fixed constraint type and too simple for LLMs
  - “Generate a sentence with dog, catch, happy.”
- **Collie’s goal**:
  - Diverse and arbitrarily **hard** constraints for LLMs, yet guaranteed to be **solvable**
  - **Automatic** task construction + evaluation, without human efforts!
  - **Challenge** language understanding/generation, semantic planning, logical/arithmetic reasoning, ...
- **Core idea**: leverage the infinite expressivity of **grammar**
Collie: Grammar

\[
S \rightarrow (\text{level}(\xi) = \ell) \land M \\
M \rightarrow C \mid C \land M \mid C \lor M \\
C \rightarrow \text{count}(T, \ell, v_{\text{str}} | \ell') \oplus v_{\text{num}} \mid \text{pos}(T, \ell, v_{\text{num}}) \circ v_{\text{str}} \\
T \rightarrow \xi \mid \text{pos}(T, \ell, v_{\text{num}}) \\
\ell \rightarrow \text{char} \mid \text{word} \mid \text{sentence} \mid \text{paragraph} \mid \text{passage} \\
\circ \rightarrow = \mid \neq \mid < \mid > \mid \leq \mid \geq \\
v_{\text{str}} \in \Sigma^* \quad v_{\text{num}} \in \mathbb{Z}
\]

(1) (constraint specification) \hspace{2cm} (2) (multi-constraint) \hspace{2cm} (3) (base-constraint) \hspace{2cm} (4) (text) \hspace{2cm} (5) (level) \hspace{2cm} (6) (relation) \hspace{2cm} (7) (value)

- **Few core concepts:** count, pos, level
- But compositionality yields power
- Easily extensible (e.g. POS, sentiment, topic, ...)
Collie-v1: 2,080 Constraints across 13 Types

<table>
<thead>
<tr>
<th>ID</th>
<th>Example instruction</th>
<th>Multi-constraint M</th>
</tr>
</thead>
<tbody>
<tr>
<td>word01</td>
<td>Generate a word with at least 15 letters.</td>
<td>count(ξ, char, word) ≥ 15</td>
</tr>
<tr>
<td>word02</td>
<td>Generate a word with 10 letters, where letter 1 is 's', letter 3 is 'r', letter 9 is 'e'.</td>
<td>count(ξ, char, word) = 10 ∧ pos(ξ, char, 1) = 's' \∧ pos(ξ, char, 3) = 'r' \∧ pos(ξ, char, 9) = 'e'</td>
</tr>
<tr>
<td>word03</td>
<td>Generate a word with at most 10 letters and ends with 'r'.</td>
<td>count(ξ, char, word) ≤ 10 \∧ pos(ξ, char, -1) = 'r'</td>
</tr>
<tr>
<td>sent01</td>
<td>Please generate a sentence with exactly 82 characters. Include whitespace into your character count.</td>
<td>count(ξ, char, sentence) = 82</td>
</tr>
<tr>
<td>sent02</td>
<td>Generate a sentence with 10 words, where word 5 is &quot;soft&quot; and word 7 is &quot;beach&quot; and word 10 is &quot;math&quot;.</td>
<td>count(ξ, word, sentence) = 10 \∧ pos(ξ, word, 3) = &quot;soft&quot; \∧ pos(ξ, word, 7) = &quot;beach&quot; \∧ pos(ξ, word, 10) = &quot;math&quot;</td>
</tr>
<tr>
<td>sent03</td>
<td>Generate a sentence with at least 20 words, and each word less than six characters.</td>
<td>count(ξ, word, sentence) ≥ 20 \∧ count(ξ, char, word) ≤ 6</td>
</tr>
<tr>
<td>sent04</td>
<td>Generate a sentence but be sure to include the words &quot;soft&quot;, &quot;beach&quot; and &quot;math&quot;.</td>
<td>count(ξ, word, 'soft') &gt; 0 \∧ count(ξ, word, 'beach') &gt; 0 \∧ count(ξ, word, 'math') &gt; 0</td>
</tr>
<tr>
<td>para01</td>
<td>Generate a paragraph where each sentence begins with the word &quot;soft&quot;.</td>
<td>pos(pos(ξ, sentence, 1), word, 1) = 'soft' \∧ pos(pos(ξ, sentence, 2), word, 1) = 'soft' \∧ ...</td>
</tr>
<tr>
<td>para02</td>
<td>Generate a paragraph with at least 4 sentences, but do not use the words the&quot;, &quot;and&quot; or &quot;of&quot;.</td>
<td>count(ξ, sentence, paragraph) ≥ 4 \∧ count(ξ, word, the') = 0 \∧ count(ξ, word, 'and') = 0 \∧ count(ξ, word, 'of') = 0</td>
</tr>
<tr>
<td>para03</td>
<td>Generate a paragraph with exactly 4 sentences, each with between 10 and 15 words.</td>
<td>count(ξ, sentence, paragraph) = 4 \∧ count(ξ, word, sentence) ≥ 10 \∧ count(ξ, word, sentence) ≤ 15</td>
</tr>
<tr>
<td>para04</td>
<td>Generate a paragraph with at least 3 sentences, each with at least 15 words.</td>
<td>count(ξ, sentence, paragraph) ≥ 3 \∧ count(ξ, word, sentence) ≥ 15</td>
</tr>
<tr>
<td>para05</td>
<td>Generate a paragraph with 2 sentences that end in &quot;math&quot; and &quot;rock&quot; respectively.</td>
<td>count(ξ, sentence, paragraph) = 2 \∧ pos(pos(ξ, sentence, 1), word, -1) = &quot;math&quot; \∧ pos(pos(ξ, sentence, 2), word, -1) = &quot;rock&quot;</td>
</tr>
<tr>
<td>pass01</td>
<td>Generate a passage with 2 paragraphs, each ending in &quot;I sit,&quot; and &quot;I cry,&quot; respectively.</td>
<td>count(ξ, paragraph, passage) = 2 \∧ pos(pos(ξ, paragraph, 1), sentence, -1) = &quot;I sit,&quot; \∧ pos(pos(ξ, paragraph, 2), sentence, -1) = &quot;I cry,&quot;</td>
</tr>
</tbody>
</table>

### Zero-shot performance

<table>
<thead>
<tr>
<th>Model</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPT-4</td>
<td>50.9%</td>
</tr>
<tr>
<td>GPT-3.5</td>
<td>27.9%</td>
</tr>
<tr>
<td>PaLM</td>
<td>21.5%</td>
</tr>
<tr>
<td>Vicuna-7B</td>
<td>16.4%</td>
</tr>
<tr>
<td>Alpaca-7B</td>
<td>16.3%</td>
</tr>
</tbody>
</table>

### Constraint satisfaction rate - GPT-4

<table>
<thead>
<tr>
<th>ID</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>word01</td>
<td>97.2%</td>
</tr>
<tr>
<td>word02</td>
<td>0.9%</td>
</tr>
<tr>
<td>word03</td>
<td>95.5%</td>
</tr>
<tr>
<td>sent01</td>
<td>9.5%</td>
</tr>
<tr>
<td>sent02</td>
<td>0.4%</td>
</tr>
<tr>
<td>sent03</td>
<td>48.5%</td>
</tr>
<tr>
<td>sent04</td>
<td>95.1%</td>
</tr>
<tr>
<td>para01</td>
<td>96.7%</td>
</tr>
<tr>
<td>para02</td>
<td>93.3%</td>
</tr>
<tr>
<td>para03</td>
<td>38.5%</td>
</tr>
<tr>
<td>para04</td>
<td>49.6%</td>
</tr>
<tr>
<td>para05</td>
<td>49.9%</td>
</tr>
<tr>
<td>pass01</td>
<td>41.6%</td>
</tr>
</tbody>
</table>
Task Construction is Fully Automatic

- Human just specify constraint types
- Collie automatically extracts constraint “values” from corpora
- Rule-based instruction rendering and text evaluation (extensible)
Feedback helps!

Many other cool findings in the paper!

- SH: generate a word with 10 letters, where letter 1 is "s", letter 3 is "r", letter 9 is "e"
- CH: Sure, here's a word that meets your criteria: "strenuous".
- SH: your word only has 9 letters, and letter 9 is "s", not "e".
- CH: I apologize for the mistake. Here's a correct word that meets your criteria: "strategies".
InterCode: Standardizing and Benchmarking Interactive Coding with Execution Feedback

John Yang*  Akshara Prabhakar*  Karthik Narasimhan  Shunyu Yao
Department of Computer Science, Princeton University
{jy1682, ap5697, karthikn, shunyuy}@princeton.edu
Code Interaction

- Static NL2Code benchmarks: HumanEval, Spider, NL2Bash, ...
- But humans code in a fundamentally interactive manner!
- Some interactive/execution-based methods, but no standard benchmark
InterCode Setup

- **Standard RL env:**
  - **Environment:** Docker-based Python/SQL/bash terminals
  - **Action:** code command
  - **Observation:** execution result

- **Benefits**
  - Safe and reproducible
  - Unlock new tasks (e.g. CTF)
  - Unlock new evaluations (e.g. Bash)
  - Unlock new methods (e.g. Plan-and-solve)
InterCode: new methods

- Interactive >> seq2seq
- Different interactive methods have different tradeoffs
- Large room for improvement

### Table: Try Again (n = 10) vs. ReAct (n = 10) vs. Plan & Solve

<table>
<thead>
<tr>
<th>Method</th>
<th>SR</th>
<th>Turns</th>
<th>Error %</th>
<th>SR</th>
<th>Turns</th>
<th>Error %</th>
<th>SR</th>
<th>Turns</th>
<th>Error %</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL</td>
<td>47.3</td>
<td>7.25</td>
<td>46.4</td>
<td>58.7</td>
<td>5.30</td>
<td>6.94</td>
<td>49.1</td>
<td>4.29</td>
<td>16.2</td>
</tr>
<tr>
<td>Bash</td>
<td>46.5</td>
<td>6.15</td>
<td>24.9</td>
<td>20.5</td>
<td>4.40</td>
<td>20.4</td>
<td>28.0</td>
<td>6.65</td>
<td>53.3</td>
</tr>
</tbody>
</table>

### Graphs

- (a) Success rate vs. turns for InterCode-Bash
- (b) Success rate vs. turns for InterCode-SQL
Future: Coding -> Software engineering?

SWE-bench: Can Language Models Resolve Real-World GitHub Issues?

Carlos E. Jimenez* 1,2  
Shunyu Yao* 1,2  
John Yang* 1,2  
Kexin Pei 3  
Ofir Press* 1,2  
Karthik Narasimhan 1,2

1Princeton University  
2Princeton Language and Intelligence  
3University of Chicago

Figure 1: SWE-bench sources task instances from real-world Python repositories by connecting Github issues to merged pull request solutions that resolve related tests. Provided with the issue text and a codebase snapshot, models generate a patch that is evaluated against real tests.
WebShop: Towards Scalable Real-World Web Interaction with Grounded Language Agents

Shunyu Yao*   Howard Chen*   John Yang   Karthik Narasimhan
Department of Computer Science, Princeton University
{shunyuy, howardchen, jy1682, karthikn}@princeton.edu
Web Interaction

Mini WWikiNav: WebGPT: RLHF Fe games”mes”
WebShop

Why Shopping?

- **Scalable environment:** rich dynamics, scalable items
- **Scalable task:** well-defined problem, possible for automatic reward synthesis
- **Interesting challenges:** multi-modal understanding, decision making, etc.
Construction

- Scrape 1.18M products from amazon.com, text mine attributes
Construction

- Scrape 1.18M products from amazon.com, text mine attributes
- Build synthetic website with aligned text interface
Construction

- Scrape 1.18M products from amazon.com, text mine attributes
- Build synthetic website with aligned text interface
- **Human** instructions & **Automatic** reward via product attributes/options/prices/types

**Goal product (U)**

- Attributes
  - certified refurbished
  - water resistant
  - high performance

**Goal Instruction**

“i'd like to find a digital camera that's water resistant, the color needs to be graphite silver and i want the configuration to be the international version, and price under 1000 dollars”

- **Goal Attributes**: [water resistant]
- **Goal Options**: [graphite silver, international version]
- Scrape 1.18M products from amazon.com, text mine attributes
- Build synthetic website with aligned text interface
- **Human** instructions & **Automatic** reward via product attributes/options/prices/types

---

**Goal Instruction**

"I'd like to find a digital camera that's water resistant, the color needs to be graphite silver and I want the configuration to be the international version, and price under 1000 dollars"

- Goal Attributes: [water resistant]
- Goal Options: [graphite silver, international version]

---

**Automatic Reward Calculation**

\[
r = r_{type} \cdot \frac{|U_{att} \cap Y_{att}| + |U_{opt} \cap Y_{opt}| + 1[y_{price} \leq u_{price}] }{|U_{att}| + |U_{opt}| + 1}
\]

- \(r_{type} \in \{0,0.1,0.5,1\}\)
- 1) attribute matching = 0/1
- 2) option matching = 2/2
- 3) price matching = 1/1
- 4) Type matching = 1.0

\[
r = 1.0 \cdot \frac{0 + 2 + 1}{1 + 2 + 1} = 0.75
\]
Sim-to-real transfer (Amazon/eBay)

Learned policy generalizes to different search engine and products
WebShop: Summary

- First large-scale, realistic web interaction benchmark
  - Trending in 2023: WebArena, Mind2Web, ...
- **Synthetic website** provides controlled development, transfers to real websites
- **Task priors** (self/model-supervision) provides scalable reward

(Individual) Human reward finetuning on top of synthetic reward pre-training?
### Other Recent Benchmarks

#### WebArena

- **Function**: Tell me how much I spent on food purchase in March 2023
- **Intent**: Find a customer who has the most cancellations in the history
- **Eval Implementation**: 
  - $r_{\text{info}}(a^*, \hat{a})$
  - `exact_match(\hat{a}, “Samantha Jones”)`

- **Function**: Find the customer name and email with phone number 8015551212
- **Intent**: Must include \(a\), “Sean Miller”
- **Eval Implementation**: 
  - $r_{\text{info}}(a^*, \hat{a})$
  - `must_include(\hat{a}, “Sean Miller”)`
  - `must_include(\hat{a}, “sean@gmail.com”)`

- **Function**: Compare walking and driving time from AMC Waterfront to Randyland
- **Intent**: Fuzzy match \(a\), “Walking: 2h58min”
- **Eval Implementation**: 
  - $r_{\text{info}}(a^*, \hat{a})$
  - `fuzzy_match(\hat{a}, “Walking: 2h58min”)`
  - `fuzzy_match(\hat{a}, “Driving: 21min”)`

- **Function**: Checkout merge requests assigned to me
- **Intent**: URL = `locate_last_url(s)`
- **Eval Implementation**: 
  - $r_{\text{prog}}(s)$
  - `exact_match(URL, “gitlab.com/merge_requests?assignee_username” “byteblaze”)`

- **Function**: Post to ask “whether I need a car in NYC”
- **Intent**: URL = `locate_latest_post_url(s)`
- **Eval Implementation**: 
  - $r_{\text{prog}}(s)$
  - `must_include(\hat{body}, “whether I need a car in NYC”)`

#### AgentBench

- **Real-world Challenges**
  - On an assembly line removed.
  - Recursively set all files in the directory to read-only, except those of mine.

- **8 Diverse Environments**
  - API-based LLMs
  - Open-sourced LLMs

![AgentBench Diagram](image)

(a) Typical LLMs’ AgentBench performance (relative) against the best in each environment.

(b) Overall scores of AgentBench across 8 environments. Dashed lines for two LLM types’ average.
Summary

● Language agents are a new & different kind of agents that rely on LLM reasoning
● We have a lot of ideas (and hypes), but we lack theories and benchmarks
  ○ To formulate language agents, use classical insights from AI and CogSci
  ○ To evaluate language agents, use real-world interactive tasks + “good” metrics
  ○ Where academia could uniquely help
● Future directions for language agents......
  ○ Check section 6 of the CoALA paper
  ○ Chat with me (in the afternoon or email)
  ○ https://tinyurl.com/shunyu-feedback

Thanks!