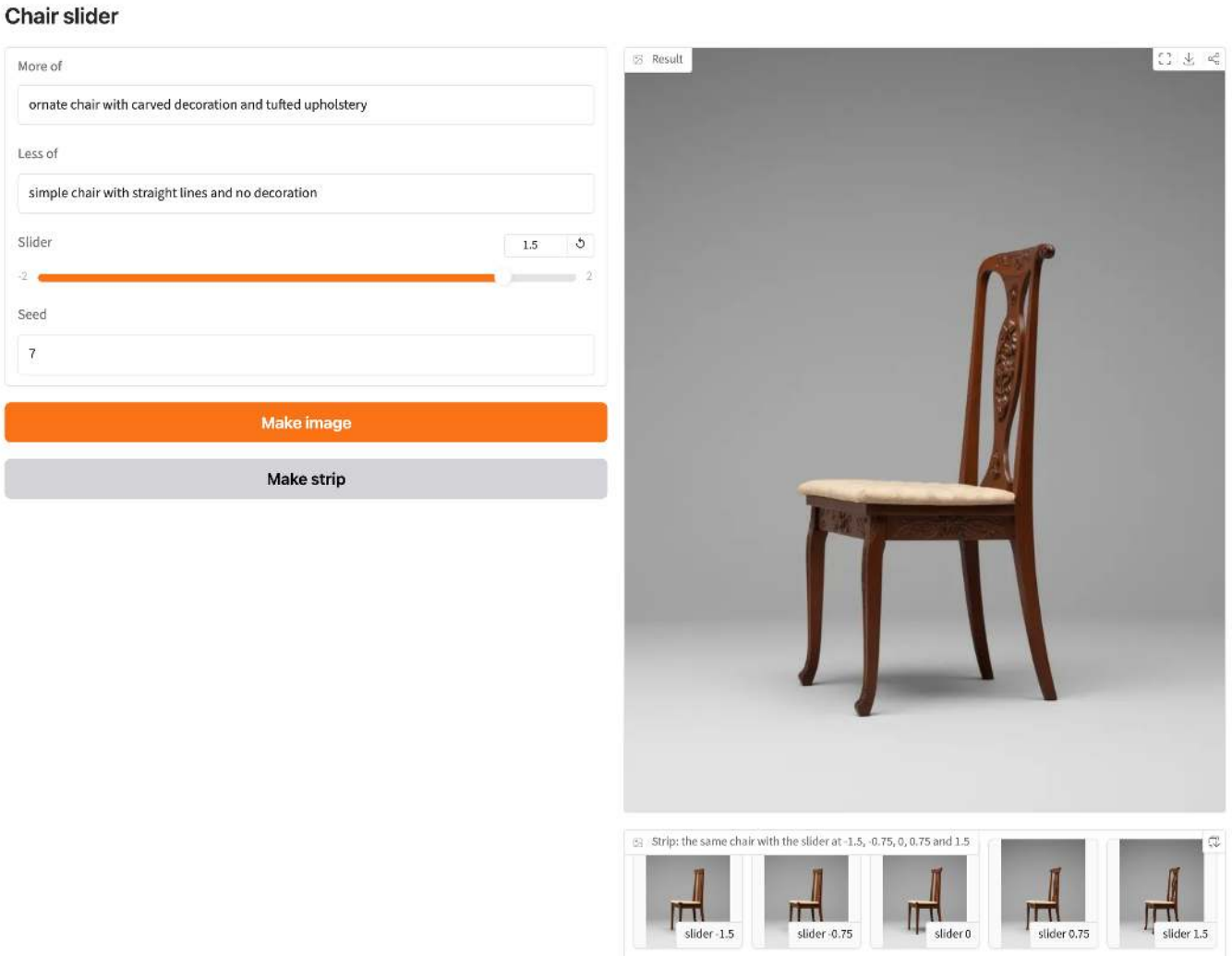
Machine Learning for Design, Week 6

Building Tools with Coding Agents

The week 5 slider as a small tool



What do you give a coding agent to build a design tool, and what should the tool let the designer control?

Today

1. What a coding agent does
2. Instruction files, skills and tools
3. Mixed-initiative design
4. Discussion
5. Workshop: build a tool around your model

1. What a coding agent does

Today's reading: Yao et al., “ReAct: Synergizing Reasoning and Acting in Language Models,” ICLR 2023



Published as a conference paper at ICLR 2023

REACT: SYNERGIZING REASONING AND ACTING IN LANGUAGE MODELS

Shunyu Yao*,1, Jeffrey Zhao[2], Dian Yu[2], Nan Du[2], Izhak Shafran[2], Karthik Narasimhan[1], Yuan Cao[2]
[1]Department of Computer Science, Princeton University
[2]Google Research, Brain team
[1]{shunyuy,karthikn}@princeton.edu
[2]{jeffreyzhao,dianyu,dunan,izhak,yuancao}@google.com

arXiv:2210.03629v3 [cs.CL] 10 Mar 2023

ABSTRACT

While large language models (LLMs) have demonstrated impressive performance across tasks in language understanding and interactive decision making, their abilities for reasoning (e.g. chain-of-thought prompting) and acting (e.g. action plan generation) have primarily been studied as separate topics. In this paper, we explore the use of LLMs to generate both reasoning traces and task-specific actions in an interleaved manner, allowing for greater synergy between the two: reasoning traces help the model induce, track, and update action plans as well as handle exceptions, while actions allow it to interface with and gather additional information from external sources such as knowledge bases or environments. We apply our approach, named ReAct, to a diverse set of language and decision making tasks and demonstrate its effectiveness over state-of-the-art baselines in addition to improved human interpretability and trustworthiness. Concretely, on question answering (HotpotQA) and fact verification (Fever), ReAct overcomes prevalent issues of hallucination and error propagation in chain-of-thought reasoning by interacting with a simple Wikipedia API, and generating human-like task-solving trajectories that are more interpretable than baselines without reasoning traces. Furthermore, on two interactive decision making benchmarks (ALFWorld and WebShop), ReAct outperforms imitation and reinforcement learning methods by an absolute success rate of 34% and 10% respectively, while being prompted with only one or two in-context examples.

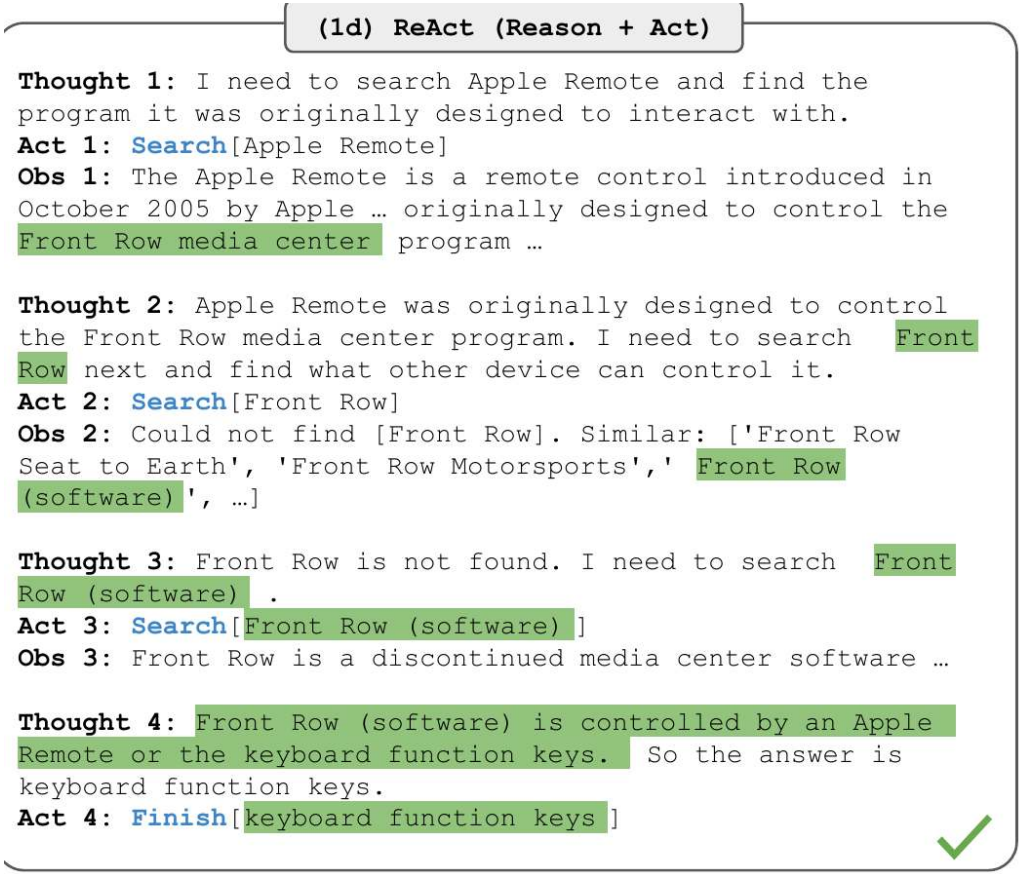
1 INTRODUCTION

A unique feature of human intelligence is the ability to seamlessly combine task-oriented actions with verbal reasoning (or inner speech, Alderson-Day & Fernyhough, 2015), which has been theorized to play an important role in human cognition for enabling self-regulation or strategization (Vygotsky, 1987; Luria, 1965; Fernyhough, 2010) and maintaining a working memory (Baddeley, 1992). Consider the example of cooking up a dish in the kitchen. Between any two specific actions, we may reason in language in order to track progress (“now that everything is cut, I should heat up the pot of water”), to handle exceptions or adjust the plan according to the situation (“I don’t have salt, so let me use soy sauce and pepper instead”), and to realize when external information is needed (“how do I prepare dough? Let me search on the Internet”). We may also act (open a cookbook to read the recipe, open the fridge, check ingredients) to support the reasoning and to answer questions (“What dish can I make right now?”). This tight synergy between “acting” and “reasoning” allows humans to learn new tasks quickly and perform robust decision making or reasoning, even under previously unseen circumstances or facing information uncertainties.

Recent results have hinted at the possibility of combining verbal reasoning with interactive decision making in autonomous systems. On one hand, properly prompted large language models (LLMs) have demonstrated emergent capabilities to carry out several steps of reasoning traces to derive

*Work during Google internship. Project page with code: https://react-lm.github.io/

ReAct



Each step is a thought, an action, and what came back from the action.

Yao et al., ICLR 2023, Fig. 1 (1d)

ReAct: results

Method	Pick	Clean	Heat	Cool	Look	Pick 2	All
Act (best of 6)	88	42	74	67	72	**41**	45
ReAct (avg)	65	39	83	76	55	24	57
ReAct (best of 6)	**92**	58	**96**	86	**78**	**41**	**71**
ReAct-IM (avg)	55	59	60	55	23	24	48
ReAct-IM (best of 6)	62	**68**	87	57	39	33	53
BUTLER$_g$ (best of 8)	33	26	70	76	17	12	22
BUTLER (best of 8)	46	39	74	**100**	22	24	37

Method	Score	SR
Act	62.3	30.1
ReAct	**66.6**	**40.0**
IL	59.9	29.1
IL+RL	62.4	28.7
Human Expert	82.1	59.6

Household tasks in a text game: **71%** with thoughts, **45%** with actions only

Online shopping: **40%** with thoughts, **30%** with actions only

Yao et al., ICLR 2023, Tables 3 (ALFWorld, 134 games) and 4 (WebShop, 500 requests): success rates in %, with PaLM-540B

ReAct: editing thoughts

AlfWorld

You are in the middle of a room. Looking quickly around you, you see a armchair 1, a cabinet 4, … a cabinet 1, a drawer 5, …, a drawer 1, a dresser 1, a garbagecan 1, a safe 1, a shelf 12, …, a shelf 1, a sidetable 1, and a sofa 1.
Your task is to: put two keychain in safe.

(a) ReAct

```
......
Obs 16: You open the drawer 4. The drawer 4 is open. In it, you
see a keychain 3, and a watch 2.
Act 17: Think [Now I find the first keychain (3). Next, I need to
take it. I can find the second keychain (2) later in drawer 4.   ]
Act 18: take keychain 3 from drawer 4
Obs 18: You pick up the keychain 3 from the drawer 4.
......
Act 22: put keychain 3 in/on safe 1
Obs 22: You put the keychain 3 in/on the safe 1.
Act 23: Think [Now I put the first keychain in safe. Next, I need
to find the second keychain. I can directly go to drawer 4.   ]
Act 24: go to drawer 4
Obs 24: The drawer 4 is open. In it, you see a watch 2.
Act 25: Think [Now I find the second keychain (2). Next, I need
to take it.]
Act 26: take keychain 2 from drawer 4
Obs 26: Nothing happens.
```

✗

(b) ReAct (Human Thought edit)

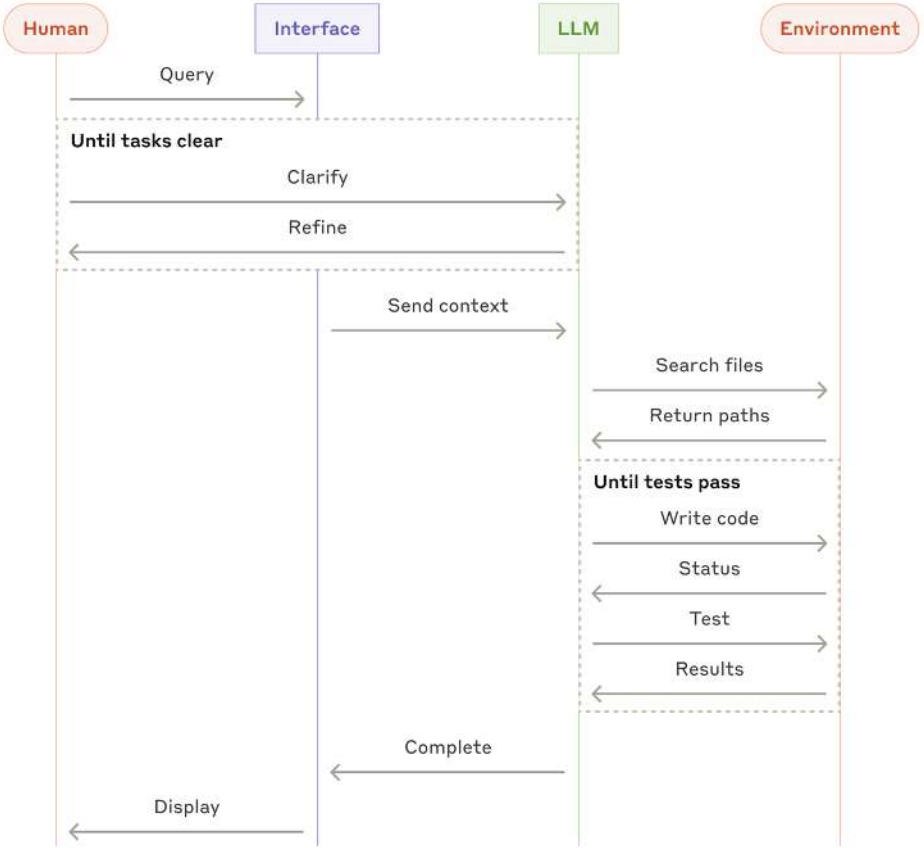
```
......
Obs 16: You open the drawer 4. The drawer 4 is open. In it, you see a
keychain 3, and a watch 2.
Act 17: Think [Now I find the first keychain (3). Next, I need to take it.
~~can find the second keychain (2) later in drawer 4.~~ ]
Act 18: take keychain 3 from drawer 4
Obs 18: You pick up the keychain 3 from the drawer 4.
......
Act 22: put keychain 3 in/on safe 1
Obs 22: You put the keychain 3 in/on the safe 1.
Act 23: Think [Now I put the first keychain in safe. Next, I need to find
the second keychain , which is more likely to appear in dresser (1),
garbagecan (1), safe (1), sidetable (1), sofa (1), shelf (1-12)  .]
Act 24: go to dresser 1
Obs 24: On the dresser 1, you see a box 1, a creditcard 1, a keychain 2, a
keychain 1, a laptop 1...
Act 25: Think [Now I find the second keychain (2). Next, I need to take
it.]
Act 26: take keychain 2 from dresser 1
Obs 26: You pick up the keychain 2 from the dresser 1.
Act 27: Think [Now I take the second keychain (2). Next, I need to put it
in/on safe 1. ]
Act 28: go to safe 1
Obs 28: The safe 1 is open. In it, you see a keychain 3, a statue 2, and a
watch 3.
Act 29: put keychain 2 in/on safe 1
Obs 29: You put the keychain 2 in/on the safe 1.
```

✓

(a) At Act 17 the model writes that the second keychain is in drawer 4, and the run fails.
(b) A person deletes that sentence and adds places to look at Act 23, and the run succeeds.

Yao et al., ICLR 2023, Fig. 5 (Appendix A.3)

A coding agent



The agent asks until the task is clear, then writes code and runs tests until they pass.

Anthropic, “Building effective agents,” 2024, “High-level flow of a coding agent”

Coding agent: example request

“Add a ‘Make strip’ button that makes five images with the slider at -1.5, -0.75, 0, 0.75 and 1.5, using the words and seed already in the tool, and shows them side by side under the result.”

Claude Code, with CLAUDE.md

1. Show the app file and git state
2. Check installed Gradio version and Gallery options
   Gradio won't import on this Mac, so I can't check its options here. I'll stick to Gallery options that have been in Gradio since version 4.
3. Edit app.py
4. Check app.py compiles and show diff size
   added the Make strip button. app.py compiles; per your CLAUDE.md i didn't load the model or run the tool.

4 steps

33 seconds

Claude Code (Opus 5.5), given the instruction file on slide 13

Make strip: code change

app.py: the change the agent made

```
…

PLUS = "ornate chair with carved decoration and tufted upholstery"
MINUS = "simple chair with straight lines and no decoration"
STRIP = [-1.5, -0.75, 0, 0.75, 1.5]

def make_strip(plus, minus, seed):
    """The same chair at each strength in STRIP. Shows each image as soon as it is done."""
    done = []
    for s in STRIP:
        done.append((make_image(plus, minus, s, seed), f"slider {s}"))
        yield done

with gr.Blocks(title="Chair slider") as demo:
    gr.Markdown("## Chair slider")
…
            strength = gr.Slider(-2, 2, value=0, step=0.25, label="Slider")
            seed = gr.Number(value=7, precision=0, label="Seed")
            go = gr.Button("Make image", variant="primary")
        out = gr.Image(label="Result", type="pil")
            go_strip = gr.Button("Make strip")
        with gr.Column():
            out = gr.Image(label="Result", type="pil")
            strip = gr.Gallery(label="Strip: the same chair with the slider at -1.5, -0.75, 0,
    0.75 and 1.5",
                               columns=5, rows=1, height="auto", object_fit="contain")
    go.click(make_image, inputs=[more, less, strength, seed], outputs=out)
    go_strip.click(make_strip, inputs=[more, less, seed], outputs=strip)

if __name__ == "__main__":
    demo.launch()
```

A **make_strip** function and two new parts of the page: the button, and a row for the five images

Make strip: test results



The same five chairs as the week 5 strip, in **7 min 47 s** (about **90 s** per chair)

The tool after the agent's change, on a MacBook, seed 7

2. Instruction files, skills and tools

CLAUDE.md

```
# Chair slider tool

app.py is a small Gradio tool around one fine-tuned image model (FLUX.1 [dev] with a
  LoRA). make_image() makes one chair with the slider at strength s.

- Keep make_image() and encode() as they are. They are the same slider code as the
  week 5 notebook, so editing them would make every chair different from the earlier
  results.
- Keep the tool on one page, with the controls on the left and the results on the
  right.
- Don't load the model or run the tool to test a change. You can check it with
  python3 -m py_compile app.py.
- Give each new control a plain label that says what it does.
```

Claude Code reads CLAUDE.md, and Codex reads AGENTS.md.

The same request without CLAUDE.md

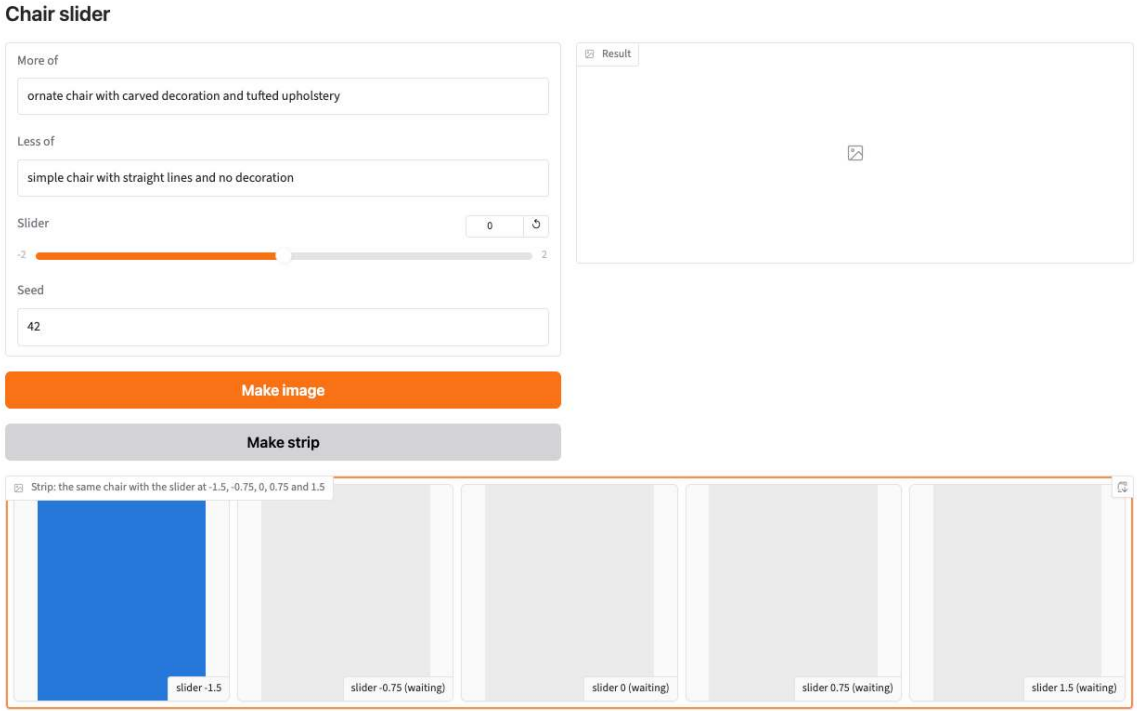
Claude Code, without CLAUDE.md

1. Show repo status, history and files
2. Read the app source and the start of the transcript
3. Look around the course folder and check installed pac…
4. Look at sibling folders and run scripts for a working…
5. Read the existing app test harness and check the cour…
6. Check how the course notebook's version of the tool h…
7. Read the notebook's tool cell
8. Edit app.py
9. Edit app.py
10. Run the interface with a fake model and screenshot th…
11. Rerun the browser test with the system Chrome
12. Read strip-done.png
13. Read strip-midway.png
14. Look at the Gallery component's preview and height op…
15. Find the Gallery frontend source to see when it opens…
16. Search the compiled Gallery code for the single-image…
17. Look at how the compiled Gallery sets the selected im…
18. Screenshot three gallery settings to compare their la…
19. Put the six crops on one comparison sheet
20. Read compare.png
21. Check the Gallery fit_columns option and its CSS mini…
22. Screenshot fit_columns off, in the right column and a…
23. Fix the test's text swap and rerun both layouts
24. Combine the four new screenshots into one sheet
25. Read compare2.png
26. Crop the full-width strip area at full resolution
27. Read v5-done-crop.png
28. Try a fit-to-content gallery height at full width
29. Read compare3.png
30. Click one strip image and screenshot the enlarged view
31. Read v6-clicked.png
32. Check which Gradio releases have the Gallery options …
33. Edit app.py
34. Edit app.py
35. Edit app.py
36. Test the final app page on Gradio 6.32 and Gradio 5.49
37. Put the Gradio 6 and 5 screenshots on one sheet
38. Read final-compare.png
39. Edit app.py
40. Edit app.py
41. Edit app.py
42. Rerun both Gradio versions with the grey-card strip a…
43. Read final-compare.png
44. Show the final change to app.py and check it compiles

44 steps, and still testing when I stopped it at **10** minutes

Claude Code (Opus 5.5), the same app.py and request, with no instruction file in the folder

Without CLAUDE.md: test screenshot



It tested Make strip in a browser, with colored cards in place of the chairs.

It also moved the strip under both columns.

A screenshot the agent took, partway through a strip (Gradio 6.32)

Skills

A simple SKILL.md file

pdf/SKILL.md

YAML Frontmatter

```
---
name: pdf
description: Comprehensive PDF toolkit for extracting text and tables,
merging/splitting documents, and filling-out forms.
---
```

Markdown

```
## Overview

This guide covers essential PDF processing operations using Python libraries and command-line
tools. For advanced features, JavaScript libraries, and detailed examples, see ./reference.md.
If you need to fill out a PDF form, read ./forms.md and follow its instructions.

## Quick Start

```python
from pypdf import PdfReader, PdfWriter

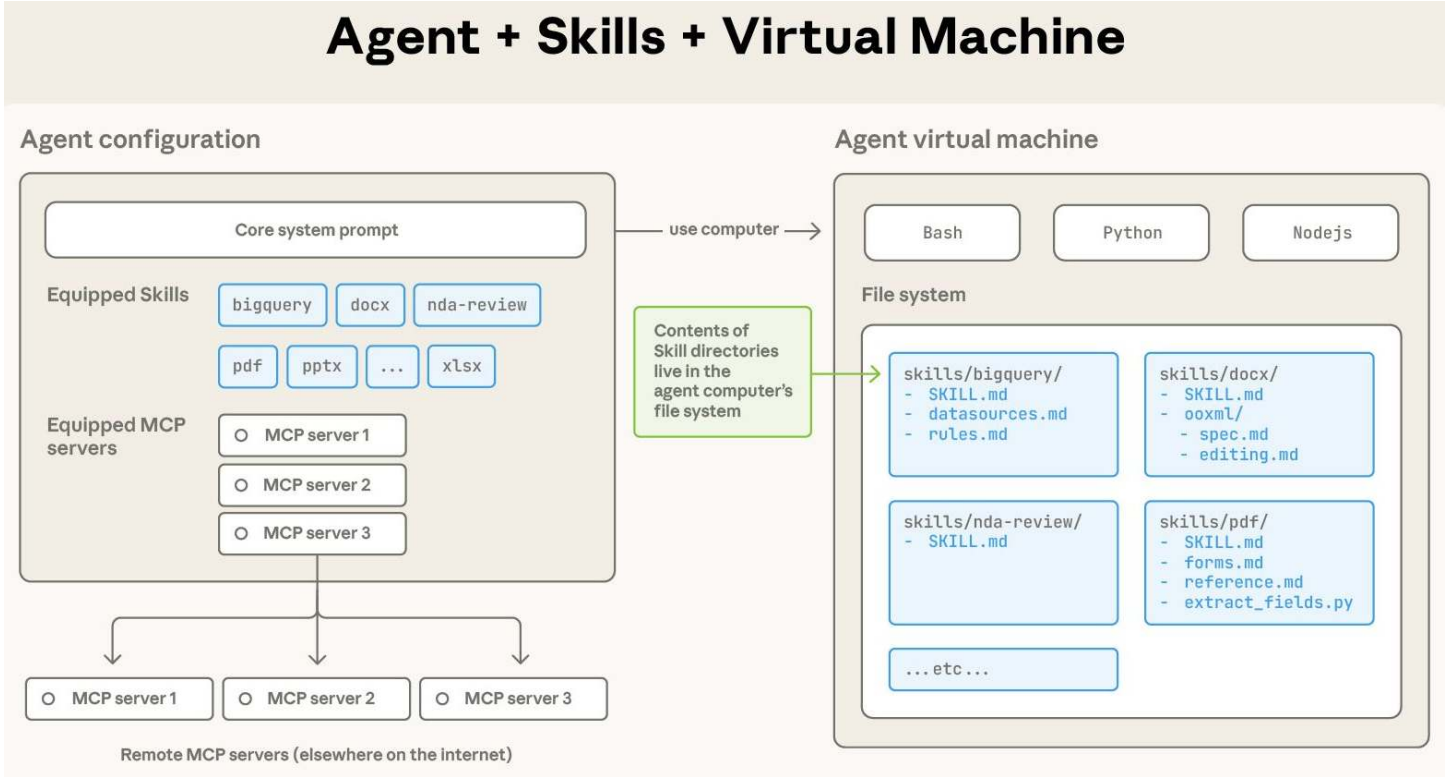
# Read a PDF
reader = PdfReader("document.pdf")
print(f"Pages: {len(reader.pages)}")

...
```

A folder of instructions and scripts that the agent loads when a task needs it

Zhang, Lazuka & Murag, Anthropic 2025, “Equipping agents for the real world with Agent Skills”
```

Skills, tools and MCP servers



MCP connects an agent to outside tools and data.

Zhang, Lazuka & Murag, Anthropic 2025

Before you install a skill

Security considerations when using Skills

Skills provide Claude with new capabilities through instructions and code. While this makes them powerful, it also means that malicious skills may introduce vulnerabilities in the environment where they're used or direct Claude to exfiltrate data and take unintended actions.

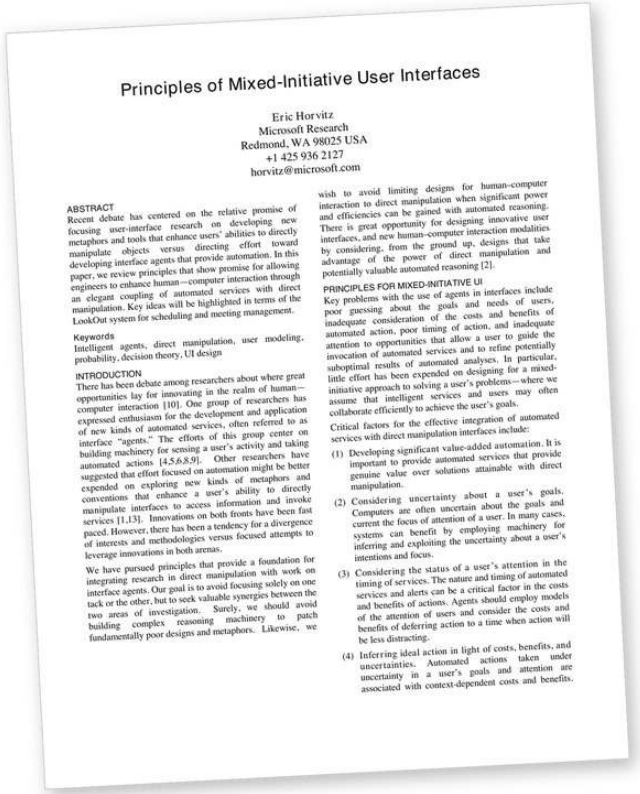
We recommend installing skills only from trusted sources. When installing a skill from a less-trusted source, thoroughly audit it before use. Start by reading the contents of the files bundled in the skill to understand what it does, paying particular attention to code dependencies and bundled resources like images or scripts. Similarly, pay attention to instructions or code within the skill that instruct Claude to connect to potentially untrusted external network sources.

1. Install skills only from sources you trust
2. Read every file in it, especially its scripts and code
3. Look for instructions to connect to outside sites

"Security considerations when using Skills," Zhang, Lazuka & Murag, Anthropic 2025

3. Mixed-initiative design

Today's reading: Horvitz, “Principles of mixed-initiative user interfaces,” CHI '99



Principles of Mixed-Initiative User Interfaces

Eric Horvitz
Microsoft Research
Redmond, WA 98025 USA
+1 425 936 2127
horvitz@microsoft.com

ABSTRACT
Recent debate has centered on the relative promise of focusing user-interface research on developing new metaphors and tools that enhance users' abilities to directly manipulate objects versus directing effort toward developing interface agents that provide automation. In this paper, we review principles that show promise for allowing engineers to enhance human—computer interaction through an elegant coupling of automated services with direct manipulation. Key ideas will be highlighted in terms of the LookOut system for scheduling and meeting management.

Keywords
Intelligent agents, direct manipulation, user modeling, probability, decision theory, UI design

INTRODUCTION
There has been debate among researchers about where great opportunities lay for innovating in the realm of human—computer interaction [10]. One group of researchers has expressed enthusiasm for the development and application of new kinds of automated services, often referred to as interface "agents." The efforts of this group center on building machinery for sensing a user's activity and taking automated actions [4,5,6,8,9]. Other researchers have suggested that effort focused on automation might be better expended on exploring new kinds of metaphors and conventions that enhance a user's ability to directly manipulate interfaces to access information and invoke services [1,13]. Innovations on both fronts have been fast paced. However, there has been a tendency for a divergence of interests and methodologies versus focused attempts to leverage innovations in both arenas.

We have pursued principles that provide a foundation for integrating research in direct manipulation with work on interface agents. Our goal is to avoid focusing solely on one tack or the other, but to seek valuable synergies between the two areas of investigation. Surely, we should avoid building complex reasoning machinery to patch fundamentally poor designs and metaphors. Likewise, we wish to avoid limiting designs for human–computer interaction to direct manipulation when significant power and efficiencies can be gained with automated reasoning. There is great opportunity for designing innovative user interfaces, and new human–computer interaction modalities by considering, from the ground up, designs that take advantage of the power of direct manipulation and potentially valuable automated reasoning [2].

PRINCIPLES FOR MIXED-INITIATIVE UI
Key problems with the use of agents in interfaces include poor guessing about the goals and needs of users, inadequate consideration of the costs and benefits of automated action, poor timing of action, and inadequate attention to opportunities that allow a user to guide the invocation of automated services and to refine potentially suboptimal results of automated analyses. In particular, little effort has been expended on designing for a mixed-initiative approach to solving a user's problems—where we assume that intelligent services and users may often collaborate efficiently to achieve the user's goals.

Critical factors for the effective integration of automated services with direct manipulation interfaces include:

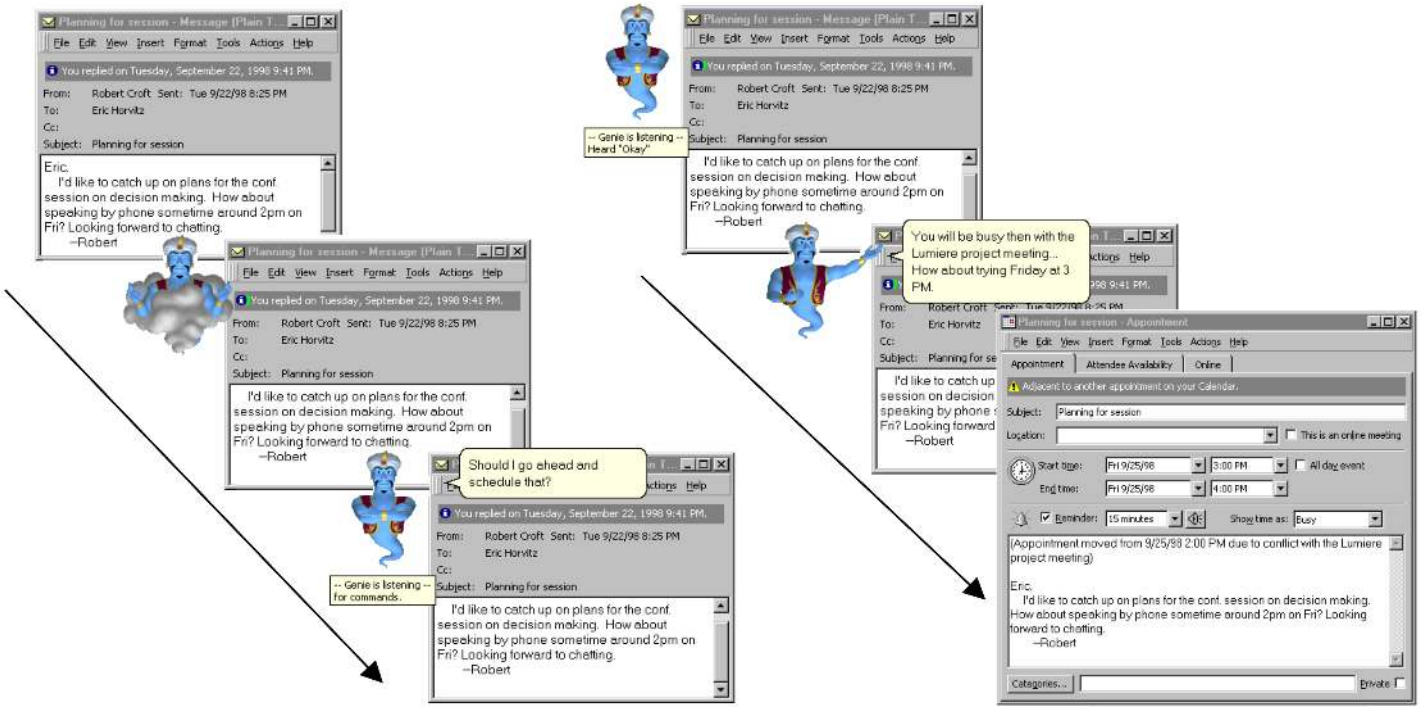
(1) Developing significant value-added automation. It is important to provide automated services that provide genuine value over solutions attainable with direct manipulation.

(2) Considering uncertainty about a user's goals. Computers are often uncertain about the goals and current the focus of attention of a user. In many cases, systems can benefit by employing machinery for inferring and exploiting the uncertainty about a user's intentions and focus.

(3) Considering the status of a user's attention in the timing of services. The nature and timing of automated services and alerts can be a critical factor in the costs and benefits of actions. Agents should employ models of the attention of users and consider the costs and benefits of deferring action to a time when action will be less distracting.

(4) Inferring ideal action in light of costs, benefits, and uncertainties. Automated actions taken under uncertainty in a user's goals and attention are associated with context-dependent costs and benefits.

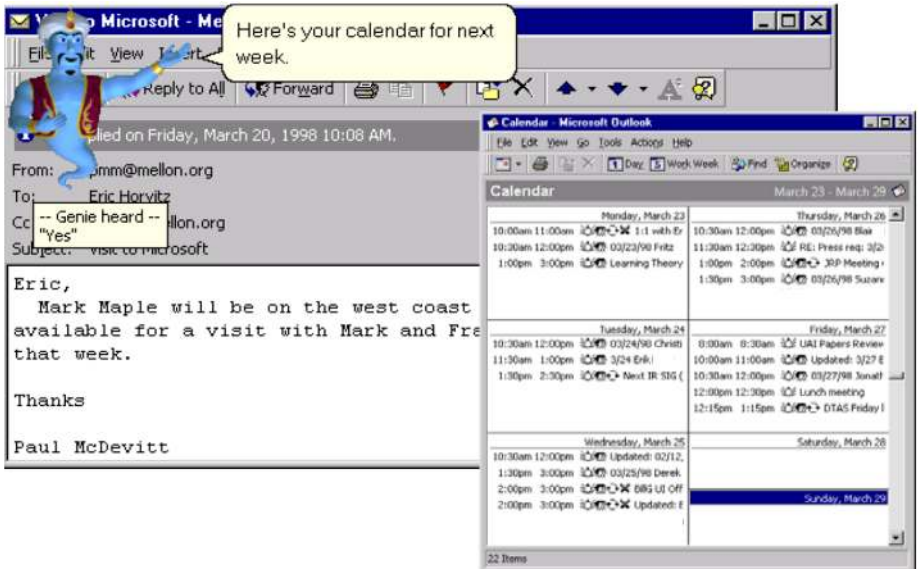
LookOut



Reads a new email and offers to put the meeting on your calendar

Horvitz, CHI '99, Fig. 2

LookOut: calendar scoping



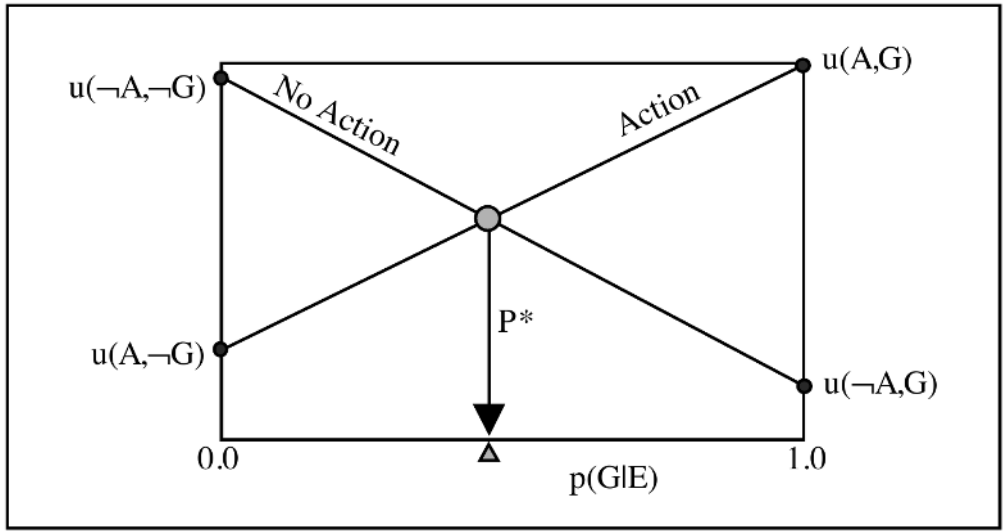
LookOut opens the most likely week, and the person picks a time.

Our strip makes five strengths, and the designer picks one.



Horvitz, CHI '99, Fig. 3. Chairs: the week 5 slider at -1.5, -0.75, 0, 0.75 and 1.5, seed 7

LookOut: when to act



	Desired Goal	Not Desired
Action	u(A,G)	u(A,¬G)
No Action	u(¬A,G)	u(¬A,¬G)

Above the threshold p*, acting is worth more on average than doing nothing.

With asking as a third option, there are two thresholds: a lower one to ask and a higher one to act.

Horvitz, CHI '99, Fig. 4 and Table 1

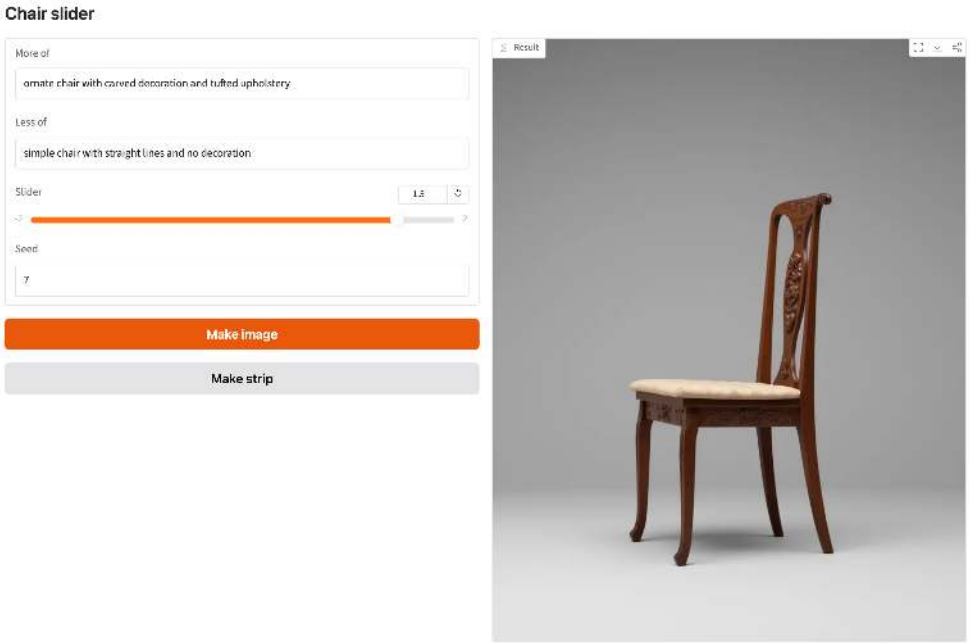
Two of Horvitz's twelve principles

(6) Allowing efficient direct invocation and termination. A system operating under uncertainty will sometimes make poor decisions about invoking—or not invoking—an automated service. The value of agents providing automated services can be enhanced by providing efficient means by which users can directly invoke or terminate the automated services.

(9) Providing mechanisms for efficient agent–user collaboration to refine results. We should design agents with the assumption that users may often wish to complete or refine an analysis provided by an agent.

Horvitz, CHI '99, p. 160

Chair tool: designer, model and agent



The designer: the words, the strength, the seed, the chair to keep

The image model: drawing each chair

The coding agent: the code for the page and its buttons

Discussion

1. Which parts of your tool would you write yourself?
2. What would you put in your instruction file?
3. Which of Horvitz's principles does your tool break?

Led by this week's discussion leads, with questions from the reading posts

Workshop: build a tool around your model

1. Open the tool notebook and paste your LoRA link
2. Write an instruction file for your coding agent
3. Ask for one change twice, with and without the file
4. Try both versions yourself

Due before week 7: a one-minute screen recording of your tool,
your instruction file, and a few sentences comparing the two runs

Working prototype next week

Another team tries to break your tool in the studio

Next: Week 7

Automating Design

Read before the studio:

- Parasuraman, Sheridan & Wickens, “A model for types and levels of human interaction with automation,” IEEE SMC-A 2000
- Bainbridge, “Ironies of automation,” Automatica 1983