Table of Contents

any to be the ultimate goal of AI. The classic book by
ial Intelligence: A Modern Approach (Prentice Hall, 1995),
study and design of rational agents."

ndation models have opened the door to agentic
aginable. These new capabilities make it finally possible
nts to act as our assistants, coworkers, and coaches.
ner data, plan a trip, do market research, manage a
y, prepare us for interviews, interview our candidates,
seem endless, and the potential economic value of these

of agents and then continue with two aspects that
tools and planning. Agents, with their new modes of
This section will end with a discussion on how to

ection of AI Engineering (2025) with minor edits to make

Notes:

1. AI-powered agents are an emerging field with no established theoretical frameworks for defining, developing, and evaluating them. This section is a best-effort attempt to build a framework from the existing literature, but it will evolve as the field does. Compared to the rest of the book, this section is more experimental. I received helpful feedback from early reviewers, and I hope to get feedback from readers of this blog post, too.
2. Just before this book came out, Anthropic published a blog post on Building effective agents (Dec 2024). I'm glad to see that Anthropic's blog post and my agent section are conceptually aligned, though with slightly different terminologies. However, Anthropic's post focuses on isolated patterns, whereas my post covers why and how things work. I also focus more on planning, tool selection, and failure modes.
3. The post contains a lot of background information. Feel free to skip ahead if it feels a little too in the weeds!

Agent Overview

Table of Contents

different engineering contexts, including but not limited
user agent, conversational agent, and reinforcement
gent?

its environment and act upon that environment.
ach (1995) defines an agent as anything that can be
hrough sensors and acting upon that environment

zed by the *environment* it operates in and *the set of*

in is defined by its use case. If an agent is developed to
that game is its environment. If you want an agent to
he environment is the internet. A self-driving car agent's
adjacent areas.

orm is augmented by the *tools* it has access to. Many
u interact with daily are agents with access to tools,
t. It can search the web, execute Python code, and
nts—text retrievers, image retrievers, and SQL executors

an agent's environment and its set of tools. The
agent can potentially use. For example, if the
environment is a chess game, the only possible actions for an agent are the valid chess moves. However, an agent's tool inventory restricts the environment it can operate in. For example, if a robot's only action is swimming, it'll be confined to a water environment.

Figure 6-8 shows a visualization of SWE-agent (Yang et al., 2024), an agent built on top of GPT-4. Its environment is the computer with the terminal and the file system. Its set of actions include navigate repo, search files, view files, and edit lines.

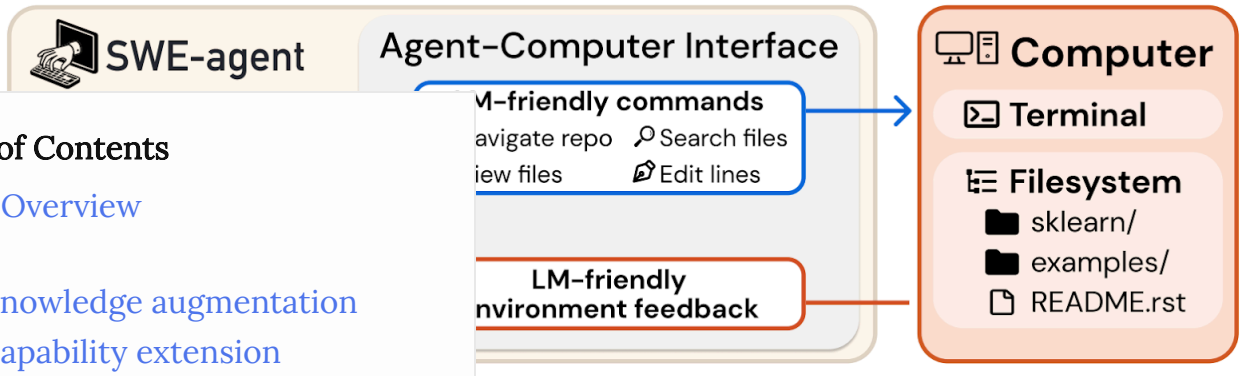


Table of Contents

agent whose environment is the computer and whose
igation, search, view files, and editing

ks typically provided by the users. In an AI agent, AI is
s a sequence of actions to achieve this task, and
accomplished.

bular data in the Kitty Vogue example above. This is a

`revenue for Fruity Fedora over the next three`
following sequence of actions:

this task. It might decide that to predict future sales, it first needs the sales numbers from the last five years. An agent's reasoning can be shown as intermediate responses.
2. Invoke SQL query generation to generate the query to get sales numbers from the last five years.
3. Invoke SQL query execution to execute this query.
4. Reason about the tool outputs (outputs from the SQL query execution) and how they help with sales prediction. It might decide that these numbers are insufficient to make a reliable projection, perhaps because of missing values. It then decides that it also needs information about past marketing campaigns.
5. Invoke SQL query generation to generate the queries for past marketing campaigns.
6. Invoke SQL query execution.
7. Reason that this new information is sufficient to help predict future sales. It then generates a projection.
8. Reason that the task has been successfully completed.

Compared to non-agent use cases, agents typically require more powerful models for two reasons:

Table of Contents

en needs to perform multiple steps to accomplish a task, s as the number of steps increases. If the model's steps, the accuracy will drop to 60%, and over 100 steps,

s, agents are capable of performing more impactful tasks, evere consequences.

ke time and money to run. A common complaint is that gh your API credits. However, if agents can be e, making their costs worthwhile.

an agent in an environment depends on **the tool it has nner**. Let's start by looking into different kinds of tools a ility for planning next.

nal tools to be an agent. However, without external tools, d. By itself, a model can typically perform one action—an enerator can generate images. External tools make an

ne environment and act upon it. Actions that allow an *read-only actions*, whereas actions that allow an agent to act upon the environment are *write actions*.

The set of tools an agent has access to is its tool inventory. Since an agent's tool inventory determines what an agent can do, it's important to think through what and how many tools to give an agent. More tools give an agent more capabilities. However, the more tools there are, the more challenging it is to understand and utilize them well. Experimentation is necessary to find the right set of tools, as discussed later in the "**Tool selection**" section.

Depending on the agent's environment, there are many possible tools. Here are three categories of tools that you might want to consider: knowledge augmentation (i.e., context construction), capability extension, and tools that let your agent act upon its environment.

Knowledge augmentation

I hope that this book, so far, has convinced you of the importance of having the relevant context for a model's response quality. An important category of tools includes those that help

augment the knowledge of your agent. Some of them have already been discussed: text retriever, image retriever, and SQL executor. Other potential tools include internal people [illegible] he status of different products, Slack retrieval, an email

Table of Contents

[illegible] your organization's private processes and information. [illegible]cess to public information, especially from the internet.

[illegible] and most anticipated capabilities to be incorporated into [illegible]odel from going stale. A model goes stale when the data it [illegible]e model's training data was cut off last week, it won't be [illegible]nformation from this week unless this information is [illegible]rowsing, a model won't be able to tell you about the [illegible]k prices, flight status, etc.

[illegible]n to cover all tools that access the internet, including [illegible] APIs, news APIs, GitHub APIs, or social media APIs.

[illegible] to reference up-to-date information to generate better [illegible]t can also open up your agent to the cesspools of the [illegible]n care.

[illegible]ress the inherent limitations of AI models. They are easy [illegible]e boost. For example, AI models are notorious for being [illegible] 199,999 divided by 292, the model will likely fail. [illegible]vial if the model had access to a calculator. Instead of trying to train the model to be good at arithmetic, it's a lot more resource-efficient to just give the model access to a tool.

Other simple tools that can significantly boost a model's capability include a calendar, timezone converter, unit converter (e.g., from lbs to kg), and translator that can translate to and from the languages that the model isn't good at.

More complex but powerful tools are code interpreters. Instead of training a model to understand code, you can give it access to a code interpreter to execute a piece of code, return the results, or analyze the code's failures. This capability lets your agents act as coding assistants, data analysts, and even research assistants that can write code to run experiments and report results. However, automated code execution comes with the risk of code injection attacks, as discussed in Chapter 5 in the section "Defensive Prompt Engineering". Proper security measurements are crucial to keep you and your users safe.

Tools can turn a text-only or image-only model into a multimodal model. For example, a model that can generate only texts can leverage a text-to-image model as a tool, allowing it to a text request, the agent's AI planner decides whether to on, or both. This is how ChatGPT can generate both text ge generator.

to generate charts and graphs, a LaTex compiler to render web pages from HTML code.

y text inputs can use an image captioning tool to process cess audio. It can use an OCR (optical character

el's performance compared to just prompting or even shows that a GPT-4-powered agent, augmented with a alone on several benchmarks. Examples of tools this query generator, an image captioner, a text detector, and

wering benchmark, Chameleon improves the best n TabMWP (Tabular Math Word Problems) (Lu et al., math questions, Chameleon improves the accuracy by

ns that allow a model to read from its data sources. But making changes to the data sources. An SQL executor can retrieve a data table (read) and change or delete the table (write). An email API can read an email but can also respond to it. A banking API can retrieve your current balance, but can also initiate a bank transfer.

Write actions enable a system to do more. They can enable you to automate the whole customer outreach workflow: researching potential customers, finding their contacts, drafting emails, sending first emails, reading responses, following up, extracting orders, updating your databases with new orders, etc.

However, the prospect of giving AI the ability to automatically alter our lives is frightening. Just as you shouldn't give an intern the authority to delete your production database, you shouldn't allow an unreliable AI to initiate bank transfers. Trust in the system's capabilities and its security measures is crucial. You need to ensure that the system is protected from bad actors who might try to manipulate it into performing harmful actions.

Table of Contents

> *Sidebar: Agents and security*

Whenever I talk about autonomous AI agents to a group of people, there is often someone who asks, "*What if someone hacks into the car to kidnap you?*" While the self-driving car example is a bit dramatic, because of its physicality, an AI system can cause harm in the real world. It can manipulate the stock market, steal copyrights, and spread misinformation and propaganda, and more, as discussed in "Prompt Engineering" in Chapter 5.

Any organization that wants to leverage AI needs to take safety seriously. This doesn't mean that AI systems should never be given the chance. If we can trust a machine to take us into space, I hope that one day it'll be safe for us to trust autonomous AI systems. Besides, humans can also trust a self-driving car more than the average stranger driver.

Table of Contents

s be vastly more productive—can you imagine doing tyscraper without cranes?—tools enable models to odel providers already support tool use with their calling. Going forward, I would expect function calling with most models.

nt is the model responsible for solving user-provided constraints. For example, one task is to schedule a two-week trip from San Francisco to India with a budget of $5,000. The goal is the two-week trip. The constraint is the budget.

Complex tasks require planning. The output of the planning process is a plan, which is a roadmap outlining the steps needed to accomplish a task. Effective planning typically requires the model to understand the task, consider different options to achieve this task, and choose the most promising one.

If you've ever been in any planning meeting, you know that planning is hard. As an important computational problem, planning is well studied and would require several volumes to cover. I'll only be able to cover the surface here.

Planning overview

Given a task, there are many possible ways to solve it, but not all of them will lead to a successful outcome. Among the correct solutions, some are more efficient than others.

Table of Contents

`ies without revenue have raised at least $1`

ions:

ue, then filter them by the amount raised.
d at least $1 billion, then filter them by revenue.

here are vastly more companies without revenue than Given only these two options, an intelligent agent should

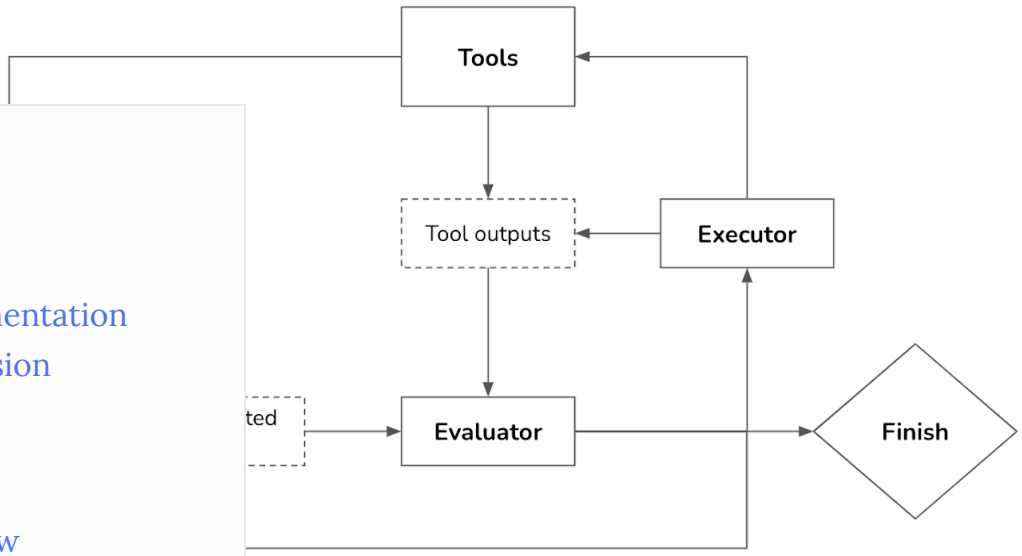
n in the same prompt. For example, you give the model a ch as with a chain-of-thought prompt), and then execute t if the model comes up with a 1,000-step plan that hout oversight, an agent can run those steps for hours, before you realize that it's not going anywhere.

should be decoupled from *execution*. You ask the agent this plan is *validated* is it executed. The plan can be , one simple heuristic is to eliminate plans with invalid a Google search and the agent doesn't have access to ther simple heuristic might be eliminating all plans with

dges. You can ask a model to evaluate whether the plan .

If the generated plan is evaluated to be bad, you can ask the planner to generate another plan. If the generated plan is good, execute it.

If the plan consists of external tools, function calling will be invoked. Outputs from executing this plan will then again need to be evaluated. Note that the generated plan doesn't have to be an end-to-end plan for the whole task. It can be a small plan for a subtask. The whole process looks like Figure 6-9.



nd execution so that only validated plans are executed

ts: one to generate plans, one to validate plans, and
ler each component an agent, this can be considered a
ntic workflows are sufficiently complex to involve
multi-agent.

nerating plans sequentially, you can generate several
o pick the most promising one. This is another latency–
ans simultaneously will incur extra costs.

ntention behind a task: what's the user trying to do with
used to help agents plan. As shown in Chapter 5 in the
pler subtasks", intent classification can be done using another prompt or a classification model trained for this task. The intent classification mechanism can be considered another agent in your multi-agent system.

Knowing the intent can help the agent pick the right tools. For example, for customer support, if the query is about billing, the agent might need access to a tool to retrieve a user's recent payments. But if the query is about how to reset a password, the agent might need to access documentation retrieval.

> *Tip:*
>
> *Some queries might be out of the scope of the agent. The intent classifier should be able to classify requests as* `IRRELEVANT` *so that the agent can politely reject those instead of wasting FLOPs coming up with impossible solutions.*

So far, we've assumed that the agent automates all three stages: generating plans, validating plans, and executing plans. In reality, humans can be involved at any stage to aid with the

Table of Contents

n, validate a plan, or execute parts of a plan. For example, ent has trouble generating the whole plan, a human an that the agent can expand upon.
such as updating a database or merging a code change, man approval before executing or defer to humans to e this possible, you need to clearly define the level of each action.

involves the following processes. Note that reflection gnificantly boost the agent's performance.

lan for accomplishing this task. A plan is a sequence of ss is also called task decomposition.
valuate the generated plan. If it's a bad plan, generate a

n the generated plan. This often involves calling specific

pon receiving the action outcomes, evaluate these the goal has been accomplished. Identify and correct ted, generate a new plan.

or plan generation and reflection in this book. When you 're asking it to decompose a task. When you ask a model to "verify if your answer is correct", you're asking it to reflect.

Foundation models as planners

An open question is how well foundation models can plan. Many researchers believe that foundation models, at least those built on top of autoregressive language models, cannot. Meta's Chief AI Scientist Yann LeCun states unequivocably that autoregressive LLMs can't plan (2023).

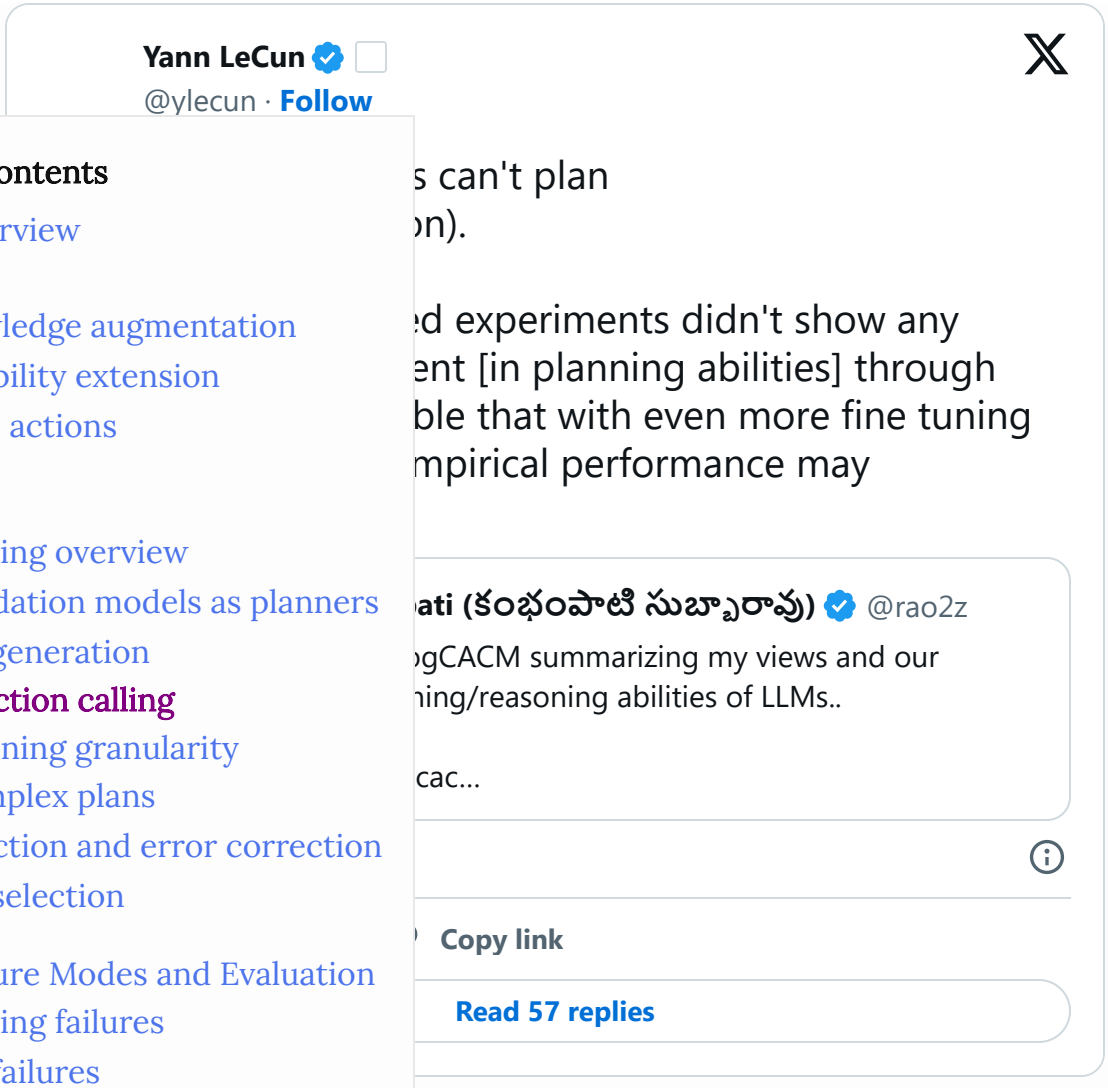


Table of Contents

nce that LLMs are poor planners, it's unclear whether it's
Ms the right way or because LLMs, fundamentally, can't

Planning, at its core, is a search problem. You search among different paths towards the goal, predict the outcome (reward) of each path, and pick the path with the most promising outcome. Often, you might determine that no path exists that can take you to the goal.

Search often requires *backtracking*. For example, imagine you're at a step where there are two possible actions: A and B. After taking action A, you enter a state that's not promising, so you need to backtrack to the previous state to take action B.

Some people argue that an autoregressive model can only generate forward actions. It can't backtrack to generate alternate actions. Because of this, they conclude that autoregressive models can't plan. However, this isn't necessarily true. After executing a path with action A, if the model determines that this path doesn't make sense, it can revise the path using action B instead, effectively backtracking. The model can also always start over and choose another path.

It's also possible that LLMs are poor planners because they aren't given the toolings needed to plan. To plan, it's necessary to know not only the available actions but also *the potential* ample, let's say you want to walk up a mountain. Your ft, turn around, or go straight ahead. However, if turning you might not consider this action. In technical terms, an ther, and it's necessary to know the outcome state to

generate only a sequence of actions like what the echnique does isn't sufficient. The paper "Reasoning with d Model" (Hao et al., 2023) argues that an LLM, by the world, is capable of predicting the outcome of each utcome prediction to generate coherent plans.

rt of a planner. It might be possible to augment an LLM ystem to help it plan.

---

sus reinforcement learning (RL) planners

ch is defined in Wikipedia as a field "*concerned with how ns in a dynamic environment in order to maximize the*

many ways. They are both characterized by their e main difference is in how their planners work.

ned by an RL algorithm. Training this RL planner can s.

- In an FM agent, the model is the planner. This model can be prompted or finetuned to improve its planning capabilities, and generally requires less time and fewer resources.

However, there's nothing to prevent an FM agent from incorporating RL algorithms to improve its performance. I suspect that in the long run, FM agents and RL agents will merge.

Plan generation

The simplest way to turn a model into a plan generator is with prompt engineering. Imagine that you want to create an agent to help customers learn about products at Kitty Vogue. You give this agent access to three external tools: retrieve products by price, retrieve top products, and retrieve product information. Here's an example of a prompt for plan generation. This prompt is for illustration purposes only. Production prompts are likely more complex.

SYSTEM PROMPT:

Table of Contents

```
Propose a plan to solve the task. You have access to 5 actions:


end_date, num_products)
)
ol_output)


lid actions.


a"
te_query, generate_response]

product last week?"
te_query, generate_response]
```


s example:

of functions whose parameters are inferred by the agent
cture the agent control flow.
kes in the task's current history and the most recent tool
fed into the response generator. The tool output at each step is added to the task's history.

Given the user input "What's the price of the best-selling product last week", a generated plan might look like this:

1. `get_time()`
2. `fetch_top_products()`
3. `fetch_product_info()`
4. `generate_query()`
5. `generate_response()`

You might wonder, "What about the parameters needed for each function?" The exact parameters are hard to predict in advance since they are often extracted from the previous tool outputs. If the first step, `get_time()`, outputs "2030-09-13", the agent can reason that the parameters for the next step should be called with the following parameters:

```
fetch_top_products(
    start_date="2030-09-07",
```

Table of Contents

to determine the exact parameter values for a function.
average price of best-selling products?", the answers to

he user wants to look at?
ng products last week, last month, or of all time?

e to make guesses, and guesses can be wrong.

the associated parameters are generated by AI models,
s can cause the model to call an invalid function or call a
ers. Techniques for improving a model's performance in
el's planning capabilities.

lanning.

with more examples.
tools and their parameters so that the model understands

es to make them simpler, such as refactoring a complex
ions.

> - *Use a stronger model. In general, stronger models are better at planning.*
> - *Finetune a model for plan generation.*

Function calling

Many model providers offer tool use for their models, effectively turning their models into agents. A tool is a function. Invoking a tool is, therefore, often called *function calling*. Different model APIs work differently, but in general, function calling works as follows:

1. *Create a tool inventory.* Declare all the tools that you might want a model to use. Each tool is described by its execution entry point (e.g., its function name), its parameters, and its documentation (e.g., what the function does and what parameters it needs).
2. *Specify what tools the agent can use for a query.*
   Because different queries might need different tools, many APIs let you specify a list of

declared tools to be used per query. Some let you control tool use further by the following settings:

...se at least one tool.
...se any tool.
...ich tools to use.

6-10. This is written in pseudocode to make it a specific API, please refer to its documentation.

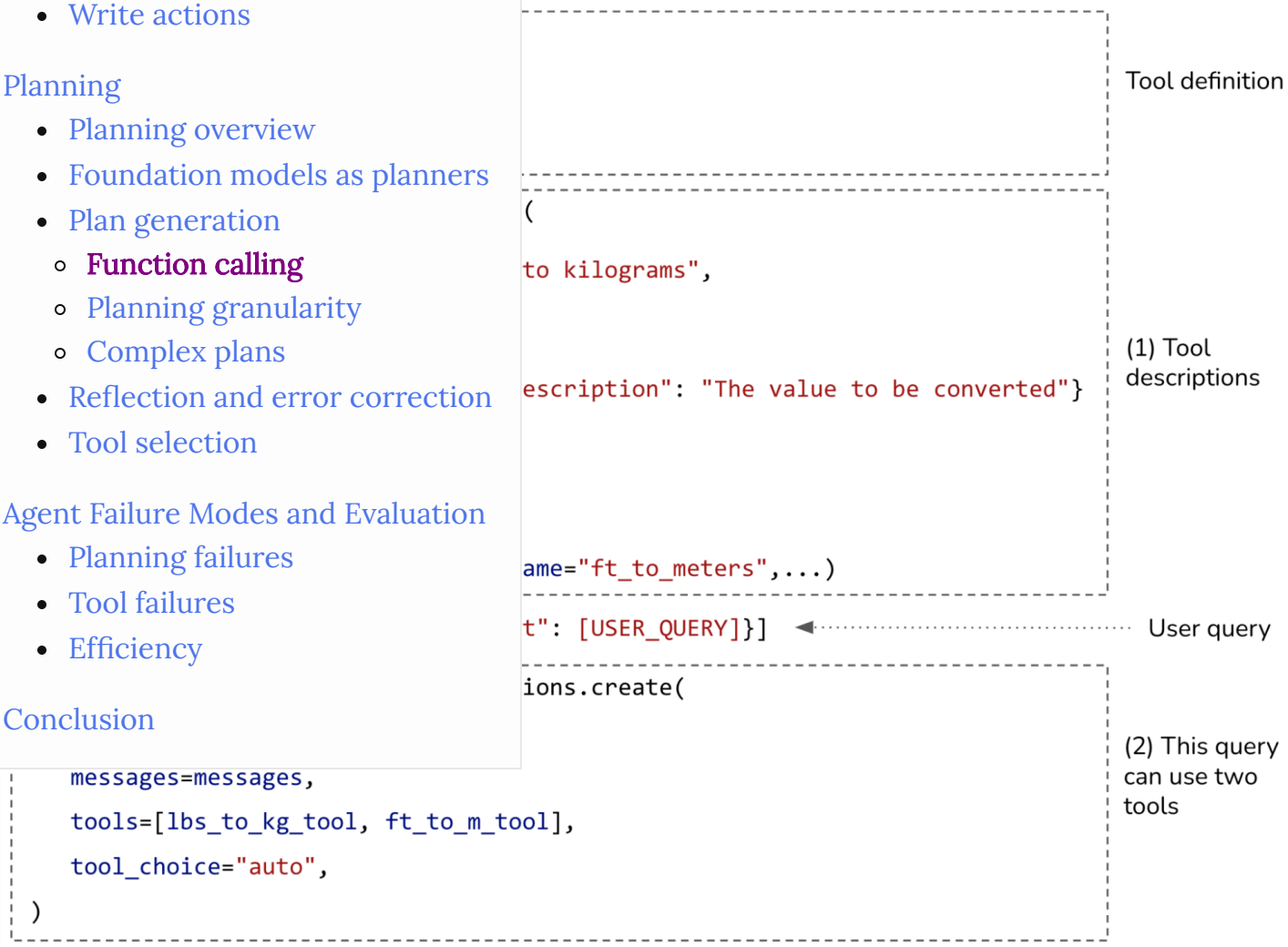


Figure 6-10. An example of a model using two simple tools

Given a query, an agent defined as in Figure 6-10 will automatically generate what tools to use and their parameters. Some function calling APIs will make sure that only valid functions are generated, though they won't be able to guarantee the correct parameter values.

For example, given the user query "How many kilograms are 40 pounds?", the agent might decide that it needs the tool `lbs_to_kg_tool` with one parameter value of 40. The agent's response might look like this.

```
response = ModelResponse(
    finish_reason='tool_calls',
```



```
(
lbs":40}',
kg'),
```

function `lbs_to_kg(lbs=40)` and use its output to

ays ask the system to report what parameter values it
hese values to make sure they are correct.

needed to accomplish a task. A roadmap can be of
or a year, a quarter-by-quarter plan is higher-level than
rn, higher-level than a week-to-week plan.

A detailed plan is harder to generate, but easier to execute. A higher-level plan is easier to generate, but harder to execute. An approach to circumvent this tradeoff is to plan hierarchically. First, use a planner to generate a high-level plan, such as a quarter-to-quarter plan. Then, for each quarter, use the same or a different planner to generate a month-to-month plan.

So far, all examples of generated plans use the exact function names, which is very granular. A problem with this approach is that an agent's tool inventory can change over time. For example, the function to get the current date `get_time()` can be renamed to `get_current_time()`. When a tool changes, you'll need to update your prompt and all your examples. Using the exact function names also makes it harder to reuse a planner across different use cases with different tool APIs.

If you've previously finetuned a model to generate plans based on the old tool inventory, you'll need to finetune the model again on the new tool inventory.

To avoid this problem, plans can also be generated using a more natural language, which is higher-level than domain-specific function names. For example, given the query "What's the eek", an agent can be instructed to output a plan that

`duct last week`

ır plan generator become robust to changes in tool APIs. atural language, it'll likely be better at understanding and nd less likely to hallucinate.

you need a translator to translate each natural language meleon (Lu et al., 2023) calls this translator a program uch simpler task than planning and can be done by llucination.

quential: the next action in the plan is *always* executed order in which actions can be executed is called a *control* pe of control flow. Other types of control flows include The list below provides an overview of each control flow,

- **Sequential**

  Executing task B after task A is complete, possibly because task B depends on task A. For example, the SQL query can only be executed after it's been translated from the natural language input.

- **Parallel**

  Executing tasks A and B at the same time. For example, given the query "Find me best-selling products under $100", an agent might first retrieve the top 100 best-selling products and, for each of these products, retrieve its price.

- **If statement**

  Executing task B or task C depending on the output from the previous step. For example, the agent first checks NVIDIA's earnings report. Based on this report, it can then decide to

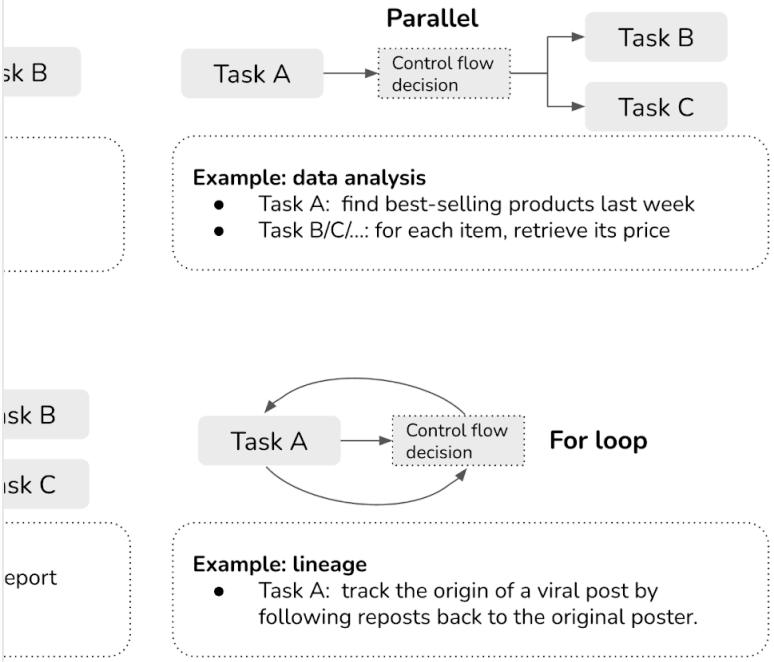
sell or buy NVIDIA stocks. Anthropic's post calls this pattern "routing".

- **For loop**

Table of Contents

ecific condition is met. For example, keep on generating
mber.

lized in Figure 6-11.



ferent orders in which a plan can be executed

nditions for control flows are exact. With AI-powered
ows. Plans with non-sequential control flows are more difficult to both generate and translate into executable commands.

> *Tip:*
>
> *When evaluating an agent framework, check what control flows it supports. For example, if the system needs to browse ten websites, can it do so simultaneously? Parallel execution can significantly reduce the latency perceived by users.*

Reflection and error correction

Even the best plans need to be constantly evaluated and adjusted to maximize their chance of success. While reflection isn't strictly necessary for an agent to operate, it's necessary for an agent to succeed.

There are many places during a task process where reflection can be useful:

- After receiving a user query to evaluate if the request is feasible.
- After the initial plan generation to evaluate whether the plan makes sense.
- ıate if it's on the right track.
- cuted to determine if the task has been accomplished.

Table of Contents

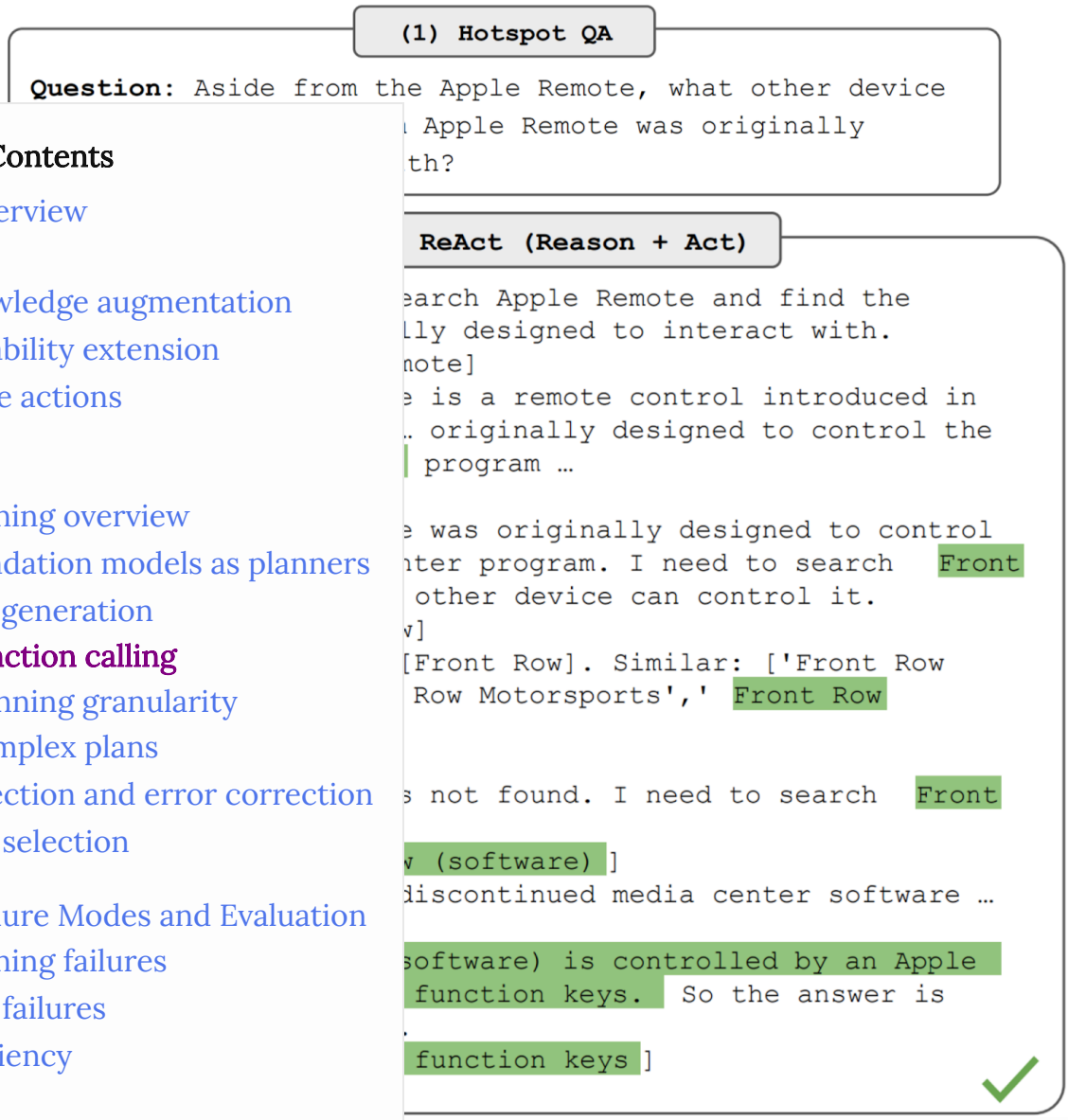
o different mechanisms that go hand in hand. Reflection
rrors to be corrected.

agent with self-critique prompts. It can also be done with
alized scorer: a model that outputs a concrete score for

2), interleaving reasoning and action has become a
used the term "reasoning" to encompass both planning
is asked to explain its thinking (planning), take actions,
until the task is considered finished by the agent. The
nples, to generate outputs in the following format:

```
rmines that the task is finished] …
```

ent following the ReAct framework responding to a question from HotpotQA (Yang et al., 2018), a benchmark for multi-hop question answering.



-12: A ReAct agent in action.

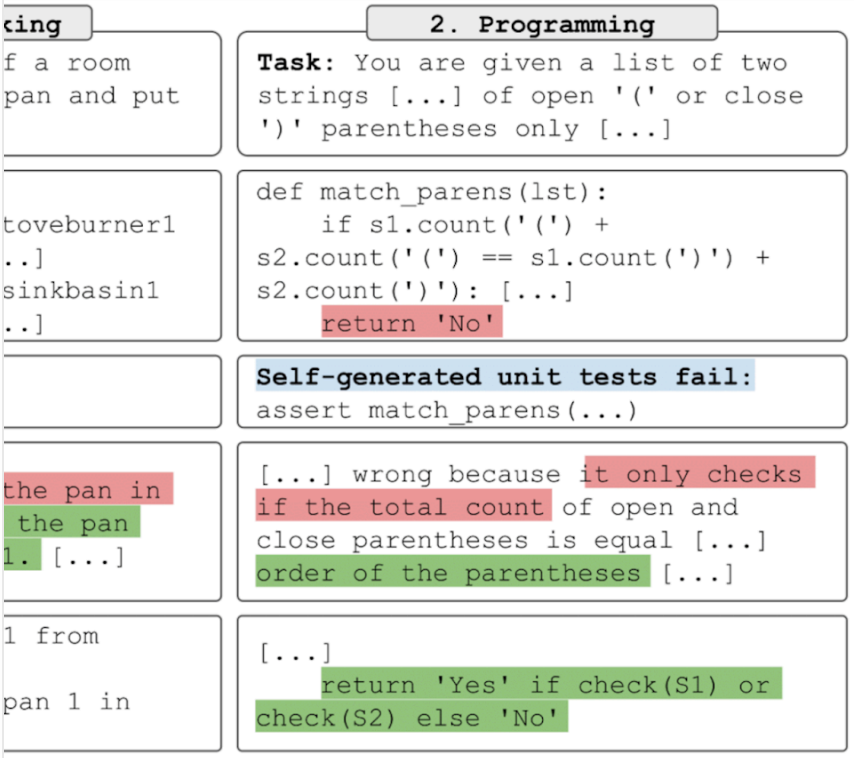
You can implement reflection in a multi-agent setting: one agent plans and takes actions and another agent evaluates the outcome after each step or after a number of steps.

If the agent's response failed to accomplish the task, you can prompt the agent to reflect on why it failed and how to improve. Based on this suggestion, the agent generates a new plan. This allows agents to learn from their mistakes.

For example, given a coding generation task, an evaluator might evaluate that the generated code fails ⅓ of the test cases. The agent then reflects that it failed because it didn't take into account arrays where all numbers are negative. The actor then generates new code, taking into account all-negative arrays.

This is the approach that Reflexion (Shinn et al., 2023) took. In this framework, reflection is separated into two modules: an evaluator that evaluates the outcome and a self-reflection

module that analyzes what went wrong. Figure 6-13 shows examples of Reflexion agents in action. The authors used the term "trajectory" to refer to a plan. At each step, after evaluation ... a new trajectory.



...nples of how Reflexion agents work.

...on is relatively easy to implement and can bring ...ement. The downside of this approach is latency and ...etimes actions can take a lot of tokens to generate, which increases cost and user-perceived latency, especially for tasks with many intermediate steps. To nudge their agents to follow the format, both ReAct and Reflexion authors used plenty of examples in their prompts. This increases the cost of computing input tokens and reduces the context space available for other information.

Tool selection

Because tools often play a crucial role in a task's success, tool selection requires careful consideration. The tools to give your agent depend on the environment and the task, but also depends on the AI model that powers the agent.

There's no foolproof guide on how to select the best set of tools. Agent literature consists of a wide range of tool inventories. For example:

- Toolformer (Schick et al., 2023) finetuned GPT-J to learn 5 tools.

- Chameleon (Lu et al., 2023) uses 13 tools.
- Gorilla (Patil et al., 2023) attempted to prompt agents to select the right API call among

lities. However, the more tools there are, the harder it is
ow it's harder for humans to master a large set of tools.
l descriptions, which might not fit into a model's

ng AI applications, tool selection requires
a few things you can do to help you decide:

with different sets of tools.
nuch the agent's performance drops if a tool is removed
removed without a performance drop, remove it.
uently makes mistakes on. If a tool proves too hard for
ensive prompting and even finetuning can't get the
he tool.
o see what tools are most used and what tools are least
ences in tool use patterns of GPT-4 and ChatGPT in

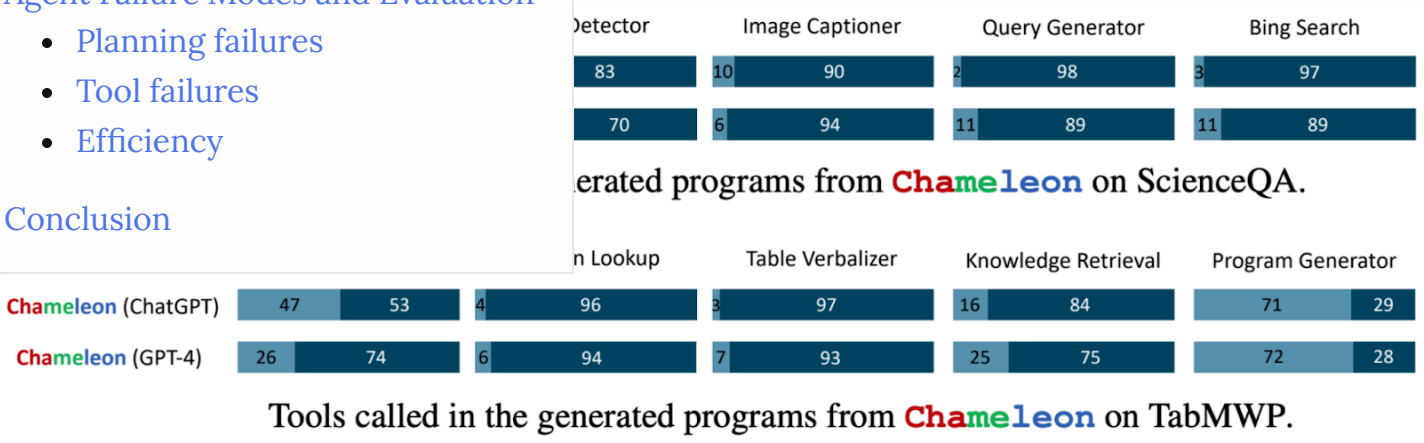


Figure 6-14. Different models and tasks express different tool use patterns.

Experiments by Chameleon (Lu et al., 2023) also demonstrate two points:

1. Different tasks require different tools. ScienceQA, the science question answering task, relies much more on knowledge retrieval tools than TabMWP, a tabular math problem-solving task.
2. Different models have different tool preferences. For example, GPT-4 seems to select a wider set of tools than ChatGPT. ChatGPT seems to favor image captioning, while GPT-4 seems to favor knowledge retrieval.

> *Tip:*
>
> *When evaluating an agent framework, evaluate what planners and tools it supports. Different planners support different categories of tools. For example, AutoGPT focuses on knowledge retrieval (Wikipedia), whereas Composio focuses on enterprise APIs (Google Suite, GitHub, etc.).*
>
> *Over time, evaluate how easy it is to extend your agent to support more tools.*

ve not just by using the tools we're given, but also by
tools from simpler ones. Can AI create new tools from its

ne study of tool transition: after tool X, how likely is the
an example of tool transition. If two tools are frequently
nto a bigger tool. If an agent is aware of this information,
ls to continually build more complex tools.

Table of Contents

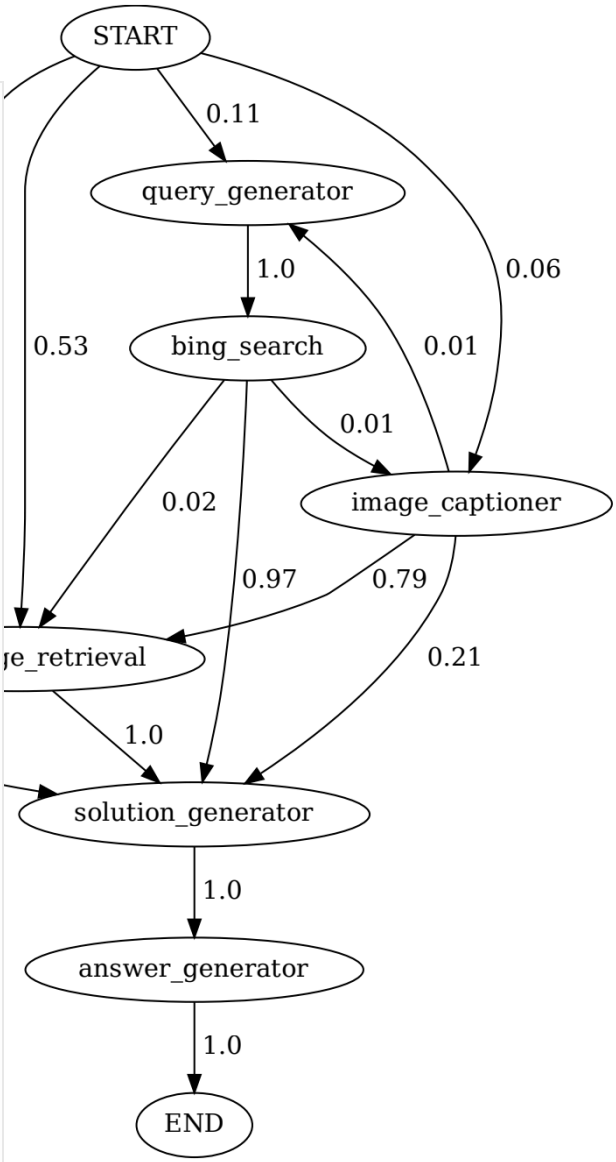


Figure 6-15. A tool transition tree by Chameleon (Lu et al., 2023).

Vogager (Wang et al., 2023) proposes a skill manager to keep track of new skills (tools) that an agent acquires for later reuse. Each skill is a coding program. When the skill manager determines a newly created skill is to be useful (e.g., because it's successfully helped an agent accomplish a task), it adds this skill to the skill library (conceptually similar to the tool inventory). This skill can be retrieved later to use for other tasks.

Earlier in this section, we mentioned that the success of an agent in an environment depends on its tool inventory and its planning capabilities. Failures in either aspect can cause the agent to fail. The next section will discuss different failure modes of an agent and how to evaluate them.

Agent Failure Modes and Evaluation

Evaluation is about detecting failures. The more complex a task an agent performs, the more possible failure points there are. Other than the failure modes common to all AI applications also have unique failures caused by planning, tool failures are easier to catch than others.

e modes and measure how often each of these failure

ays. The most common mode of planning failure is tool a plan with one or more of these errors.

nat contains `bing_search`, which isn't in the tool

with two parameters, but this function requires only one

lues

`ith` one parameter, `lbs`, but uses the value 100 for lbs

al failure: the agent fails to achieve the goal. This can be because the plan doesn't solve a task, or it solves the task without following the constraints. To illustrate this, imagine you ask the model to plan a two-week trip from San Francisco to India with a budget of $5,000. The agent might plan a trip from San Francisco to Vietnam, or plan you a two-week trip from San Francisco to India that will cost you way over the budget.

A common constraint that is often overlooked by agent evaluation is time. In many cases, the time an agent takes matters less because you can assign a task to an agent and only need to check in when it's done. However, in many cases, the agent becomes less useful with time. For example, if you ask an agent to prepare a grant proposal and the agent finishes it after the grant deadline, the agent isn't very helpful.

An interesting mode of planning failure is caused by errors in reflection. The agent is convinced that it's accomplished a task when it hasn't. For example, you ask the agent to assign 50 people to 30 hotel rooms. The agent might assign only 40 people and insist that the task has been accomplished.

Table of Contents

To evaluate an agent for planning failures, one option is to create a planning dataset where each example is a tuple `(task, tool inventory)`. For each task, use the agent to generate a K ng metrics:

any are valid?
does the agent have to generate to get a valid plan?
valid?
?
ith invalid parameters?
ith incorrect parameter values?

Table of Contents

ns. What types of tasks does the agent fail more on? Do s does the model frequently make mistakes with? Some se. You can improve an agent's ability to use a more examples, or finetuning. If all fail, you might nething easier to use.

tool is used, but the tool output is wrong. One failure ng outputs. For example, an image captioner returns a nerator returns a wrong SQL query.

plans and a translation module is involved in translating e commands, failures can happen because of translation

Tool failures are tool-dependent. Each tool needs to be tested independently. Always print out each tool call and its output so that you can inspect and evaluate them. If you have a translator, create benchmarks to evaluate it.

Detecting missing tool failures requires an understanding of what tools should be used. If your agent frequently fails on a specific domain, this might be because it lacks tools for this domain. Work with human domain experts and observe what tools they would use.

Efficiency

An agent might generate a valid plan using the right tools to accomplish a task, but it might be inefficient. Here are a few things you might want to track to evaluate an agent's efficiency:

- How many steps does the agent need, on average, to complete a task?
- How much does the agent cost, on average, to complete a task?

- How long does each action typically take? Are there any actions that are especially time-consuming or expensive?

Table of Contents

our baseline, which can be another agent or a human
o human agents, keep in mind that humans and AI have
what's considered efficient for humans might be
ample, visiting 100 web pages might be inefficient for a
ge at a time but trivial for an AI agent that can visit all

airly simple. An agent is defined by the environment it
ccess to. In an AI-powered agent, the AI model is the
ack from the environment to plan how best to
es a model vastly more capable, so the agentic pattern is

novel, they are built upon many concepts that have been
uding self-critique, chain-of-thought, and structured

gents work and different components of an agent. In a
agent frameworks.

nformation that exceeds a model's context limit. A
model's context in handling information can significantly enhance an agent's capabilities. Since this post is already long, I'll explore how a memory system works in a future blog post.

What do you think?

415 Responses

 Love

 Surprised

 Angry

 Sad

ge without restrictive paywall. Your last article "Building GenAI
e about RAGs. Explaining the concepts right from the simplest
took me on a journey on learning.

h the content and it seems to cover almost what i have been mean to read. I didn't mean to dive deep into agents, still wanted to understand how agents are being integrated into what we already know about LLMs. This would a good read over the weekend.

A suggestion, if you could include a source if someone wants to understand further than what is already mentioned.

4 0 Reply

Tobi → Prabin Kumar Nayak

a month ago

You can check out her new book: AI Engineering Building Application with Foundation Models, the book has more details, and you will further understand what is already mentioned, I am currently going through the book and I have learned a lot and would highly recommend it

2 0 Reply

Dushyant Singh
a month ago

Thanks for sharing! Means a lot

Table of Contents

. Appreciate your efforts. Could you please explain the
l AI Agent?

cle!

otential small mistake I observed, and maybe you need to edit it

ool preferences. For example, GPT-4 seems to select a wider set
eems to favor image captioning, while GPT-4 seems to favor

e ChatGPT here, as ChatGPT is not a model itself but rather the
nt models behind, and you probably meant another OpenAI

!

Narayanan Arvind
a month ago

A wonderful post on agents @Chip Huyen. Looking forward to reading your book on AI Engineering. We are working on the text2code problem and agentic AI systems look promising for us. Mathematical and image tools look interesting. Latency is of concern for us. Any suggestions on how to decrease the latency of an agentic system?

o o Reply

Lakshman K
a month ago

+1 on sharing your knowledge without restrictive paywall.

I spent few hours reading this article on a flight and this is an extremely well written, easy to understand, and more importantly of great help to anyone working with LLMs and agents. Highly recommend! I'll be getting my copy of 'AI Engineering Building Application with Foundation Models'.

o o Reply

Table of Contents

urify if any agent is expected to handle planning by itself
? Or do we consider a system with rigid sequence of actions (e.g.
ere each router label sends flow into a specific predefined
ad-hoc planning by the system itself) also to be "an agent"?

uestion I have is about the differences and definitions between :
odel agent. Could you elaborate on those?

ecently built the tools regression model and it was super similar
ed for the book for so long now, when is it releasing in India???

p companies deploy machine learning into production. I
e about AI applications, tooling, and best practices.

huyenchip19
chiphuyen

Email Address

Subscribe