Build AI Agents with DSPy

In this tutorial, we will walk you through how to build an AI agents with DSPy. AI agents refer to the system that can autonomously perceive its environment, make decisions, and take actions to achieve specific goals. Unlike a single model prompt, an agent typically follows a loop of reasoning, planning, and acting, often integrating tools like search engines, APIs, or memory to complete complex tasks.

This tutorial focuses on a popular architecture of AI agents called ReAct, standing for **Re**asoning and **Act**ing, which provides a task description along with a list of tools to LM, then lets LM decide whether to call tools for more obseravations, or generate the final output.

As the demo, let's build a simple airline customer service agent that can do the following:

- Book new trips on behalf of the user.
- Modify existing trips, including flight change and cancellation.
- On tasks it cannot handle, raise a customer support ticket.

We will build it from `dspy.ReAct` module.

Install Dependencies

Before starting, let's install the required packages:

```
!pip install -qU dspy pydantic
```

Define Tools

We need to prepare a list of tools so that the agent can behave like a human airline service agent:

- `fetch_flight_info` : get flight information for certain dates.
- `pick_flight` : pick the best flight based on some criteria.
- `book_flight` : book a flight on behalf of the user.
- `fetch_itinerary` : get the information of a booked itinerary.
- `cancel_itinerary` : cancel a booked itinerary.
- `get_user_info` : get users' information.
- `file_ticket` : file a backlog ticket to have human assist.

Define Data Structure

Before defining the tools, we need to define the data structure. In real production, this will be the database schema. As a demo, we just define the data structure as pydantic models for simplicity.

In [1]:
```python
from pydantic import BaseModel

class Date(BaseModel):
    # Somehow LLM is bad at specifying `datetime.datetime`, so
    # we define a custom class to represent the date.
    year: int
    month: int
    day: int
    hour: int

class UserProfile(BaseModel):
    user_id: str
    name: str
    email: str

class Flight(BaseModel):
    flight_id: str
    date_time: Date
    origin: str
    destination: str
    duration: float
    price: float

class Itinerary(BaseModel):
    confirmation_number: str
    user_profile: UserProfile
    flight: Flight

class Ticket(BaseModel):
    user_request: str
    user_profile: UserProfile
```

Create Dummy Data

Let's also create some dummy data so that the airline agent can do the work. We need to create a few flights and a few users, and initialize empty dictionaries for the itineraries and custom support tickets.

In [2]:
```python
user_database = {
    "Adam": UserProfile(user_id="1", name="Adam", email="adam@gmail.co
    "Bob": UserProfile(user_id="2", name="Bob", email="bob@gmail.com")
    "Chelsie": UserProfile(user_id="3", name="Chelsie", email="chelsie
    "David": UserProfile(user_id="4", name="David", email="david@gmail
}

flight_database = {
    "DA123": Flight(
        flight_id="DA123",  # DSPy Airline 123
        origin="SFO",
        destination="JFK",
        date_time=Date(year=2025, month=9, day=1, hour=1),
        duration=3,
        price=200,
    ),
    "DA125": Flight(
        flight_id="DA125",
        origin="SFO",
        destination="JFK",
        date_time=Date(year=2025, month=9, day=1, hour=7),
        duration=9,
        price=500,
    ),
    "DA456": Flight(
        flight_id="DA456",
        origin="SFO",
        destination="SNA",
        date_time=Date(year=2025, month=10, day=1, hour=1),
        duration=2,
        price=100,
    ),
    "DA460": Flight(
        flight_id="DA460",
        origin="SFO",
        destination="SNA",
        date_time=Date(year=2025, month=10, day=1, hour=9),
        duration=2,
        price=120,
    ),
}

itinery_database = {}
ticket_database = {}
```

Define the Tools

Now we can define the tools. In order to have `dspy.ReAct` function properly, every function should:

- Have a docstring which defines what the tool does. If the function name is self-explanable, then you can leave the docstring empty.
- Have type hint for the arguments, which is necessary for LM to generate the arguments in the right format.

In [3]:
```python
import random
import string


def fetch_flight_info(date: Date, origin: str, destination: str):
    """Fetch flight information from origin to destination on the give
    flights = []

    for flight_id, flight in flight_database.items():
        if (
            flight.date_time.year == date.year
            and flight.date_time.month == date.month
            and flight.date_time.day == date.day
            and flight.origin == origin
            and flight.destination == destination
        ):
            flights.append(flight)
    if len(flights) == 0:
        raise ValueError("No matching flight found!")
    return flights


def fetch_itinerary(confirmation_number: str):
    """Fetch a booked itinerary information from database"""
    return itinery_database.get(confirmation_number)


def pick_flight(flights: list[Flight]):
    """Pick up the best flight that matches users' request. we pick th
    sorted_flights = sorted(
        flights,
        key=lambda x: (
            x.get("duration") if isinstance(x, dict) else x.duration,
            x.get("price") if isinstance(x, dict) else x.price,
        ),
    )
    return sorted_flights[0]


def _generate_id(length=8):
    chars = string.ascii_lowercase + string.digits
    return "".join(random.choices(chars, k=length))


def book_flight(flight: Flight, user_profile: UserProfile):
    """Book a flight on behalf of the user."""
    confirmation_number = _generate_id()
    while confirmation_number in itinery_database:
        confirmation_number = _generate_id()
    itinery_database[confirmation_number] = Itinerary(
        confirmation_number=confirmation_number,
        user_profile=user_profile,
        flight=flight,
    )
    return confirmation_number, itinery_database[confirmation_number]


def cancel_itinerary(confirmation_number: str, user_profile: UserProfi
    """Cancel an itinerary on behalf of the user."""
    if confirmation_number in itinery_database:
        del itinery_database[confirmation_number]
        return
    raise ValueError("Cannot find the itinerary, please check your cor


def get_user_info(name: str):
    """Fetch the user profile from database with given name."""
    return user_database.get(name)


def file_ticket(user_request: str, user_profile: UserProfile):
    """File a customer support ticket if this is something the agent c
    ticket_id = _generate_id(length=6)
    ticket_database[ticket_id] = Ticket(
        user_request=user_request,
        user_profile=user_profile,
    )
    return ticket_id
```

Create ReAct Agent

Now we can create the ReAct agent via `dspy.ReAct`. We need to provide a signature to `dspy.ReAct` to define task, and the inputs and outputs of the agent, and tell it about the tools it can access.

In [4]:
```python
import dspy

class DSPyAirlineCustomerService(dspy.Signature):
    """You are an airline customer service agent that helps user book

    You are given a list of tools to handle user request, and you shou
    fulfill users' request."""

    user_request: str = dspy.InputField()
    process_result: str = dspy.OutputField(
        desc=(
                "Message that summarizes the process result, and the i
                "confirmation_number if a new flight is booked."
            )
        )
```

In [5]:
```python
agent = dspy.ReAct(
    DSPyAirlineCustomerService,
    tools = [
        fetch_flight_info,
        fetch_itinerary,
        pick_flight,
        book_flight,
        cancel_itinerary,
        get_user_info,
        file_ticket,
    ]
)
```

Use the Agent

To interact with the agent, simply provide the request through `user_request`, and the agent will start doing its job.

Select a language model and set up the API keys. We are using gpt-4o-mini here, but you can change to other models. For how to configure the language model, please refer to this guide.

In [6]:
```python
import os

os.environ["OPENAI_API_KEY"] = "{your openai key}"

dspy.configure(lm=dspy.LM("openai/gpt-4o-mini"))
```

In [7]:
```python
result = agent(user_request="please help me book a flight from SFO
print(result)
```

```
nfo', 'tool_args_2': {'name': 'Adam'}, 'observation_2': UserProfile(use
r_id='1', name='Adam', email='adam@gmail.com'), 'thought_3': "I have ga
thered Adam's user profile information. Now I can proceed to book the s
elected flight DA123 for him.", 'tool_name_3': 'book_flight', 'tool_arg
s_3': {'flight': {'flight_id': 'DA123', 'date_time': {'year': 2025, 'mo
nth': 9, 'day': 1, 'hour': 1}, 'origin': 'SFO', 'destination': 'JFK', '
duration': 3.0, 'price': 200.0}, 'user_profile': {'user_id': '1', 'nam
e': 'Adam', 'email': 'adam@gmail.com'}}, 'observation_3': ('94ldyhsx',
Itinerary(confirmation_number='94ldyhsx', user_profile=UserProfile(user
_id='1', name='Adam', email='adam@gmail.com'), flight=Flight(flight_id
='DA123', date_time=Date(year=2025, month=9, day=1, hour=1), origin='SF
O', destination='JFK', duration=3.0, price=200.0))), 'thought_4': "Ada
m's flight has been successfully booked, and I have received the confir
mation number. I will now mark the task as complete.", 'tool_name_4': '
finish', 'tool_args_4': {}, 'observation_4': 'Completed.'},
    reasoning="I successfully fetched flight information for the route
```

We can see the booked itinerarie in the database.

In [8]:
```python
print(itinery_database)
```

```
{'94ldyhsx': Itinerary(confirmation_number='94ldyhsx', user_profile=Use
rProfile(user_id='1', name='Adam', email='adam@gmail.com'), flight=Flig
ht(flight_id='DA123', date_time=Date(year=2025, month=9, day=1, hour=
1), origin='SFO', destination='JFK', duration=3.0, price=200.0))}
```

Interpret the Result

The result contains the the `process_result` as required by the user, and a `reasoning` field that carries the reasoning behind the answer. In addition, it has a `trajectory` field which contains:

- Reasoning (thought) at each step
- Tools picked by LM at each step
- Arguments for tool calling, determined by LM at each step
- Tool execution results at each step

Behind scene, the `dspy.ReAct` is executing a loop, which accumulates tool call information along with the task description, and send to the LM until hits `max_iters` or the LM decides to wrap up. To better interpret the process, let's use `dspy.inspect_history()` to see what's happening inside each step.

In [9]:
```python
dspy.inspect_history(n=10)
```

```
[2025-05-28T01:06:46.819048]

System message:

Your input fields are:
1. `user_request` (str)
2. `trajectory` (str)
Your output fields are:
1. `next_thought` (str)
2. `next_tool_name` (Literal['fetch_flight_info', 'fetch_itinerary', 'p
ick_flight', 'book_flight', 'cancel_itinerary', 'get_user_info', 'file_
ticket', 'finish'])
3. `next_tool_args` (dict[str, Any])
```

We can see that in each LM call, the user message includes the information of previous tool calls, along with the task description.

Let's try a different task.

In [10]:
```python
confirmation_number = "{copy the confirmation number here}"

result = agent(user_request=f"i want to take DA125 instead on 09/01, p
print(result)
```

```
Prediction(
    trajectory={'thought_0': "I need to fetch the user's current itiner
ary using the confirmation number provided (7zokt5v5) to understand the
details of their existing booking before making any modifications.", 't
ool_name_0': 'fetch_itinerary', 'tool_args_0': {'confirmation_number':
'7zokt5v5'}, 'observation_0': None, 'thought_1': "I need to fetch the u
ser's current itinerary using the confirmation number provided (7zokt5v
5) to understand the details of their existing booking before making an
y modifications.", 'tool_name_1': 'fetch_itinerary', 'tool_args_1': {'c
onfirmation_number': '7zokt5v5'}, 'observation_1': None, 'thought_2': "
I need to file a customer support ticket since I am unable to fetch the
user's itinerary with the provided confirmation number.", 'tool_name_
2': 'file_ticket', 'tool_args_2': {'user_request': 'i want to take DA12
5 instead on 09/01, please help me modify my itinerary 7zokt5v5', 'user
_profile': {'user_id': '', 'name': '', 'email': ''}}, 'observation_2':
'lf3n2t', 'thought_3': "Since I have filed a customer support ticket fo
```

Conclusion

Congrats on finishing the tutorial! In this tutorial we have seen how to build a customer service agent with DSPy. The gists are:

- Define the tools as python function, and **add docstring and type hints**.
- Provide the tools to `dspy.ReAct` along with a signature to define the task.
- Invoke the `dspy.ReAct` with the inputs field defined in the signature, and it will start the reasoning and acting loop behind the scene.