

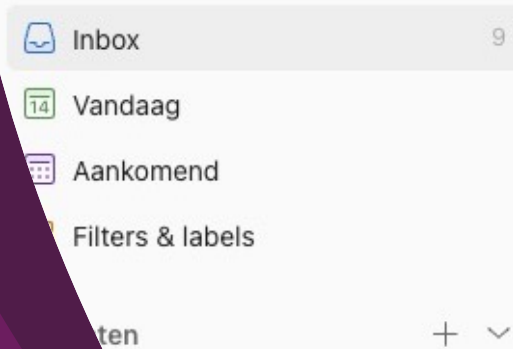


LLM Agents 101

How I Gave ChatGPT Access to My To-Do List

By Jordi Smit

Machine learning engineer at Xebia Data



Inbox



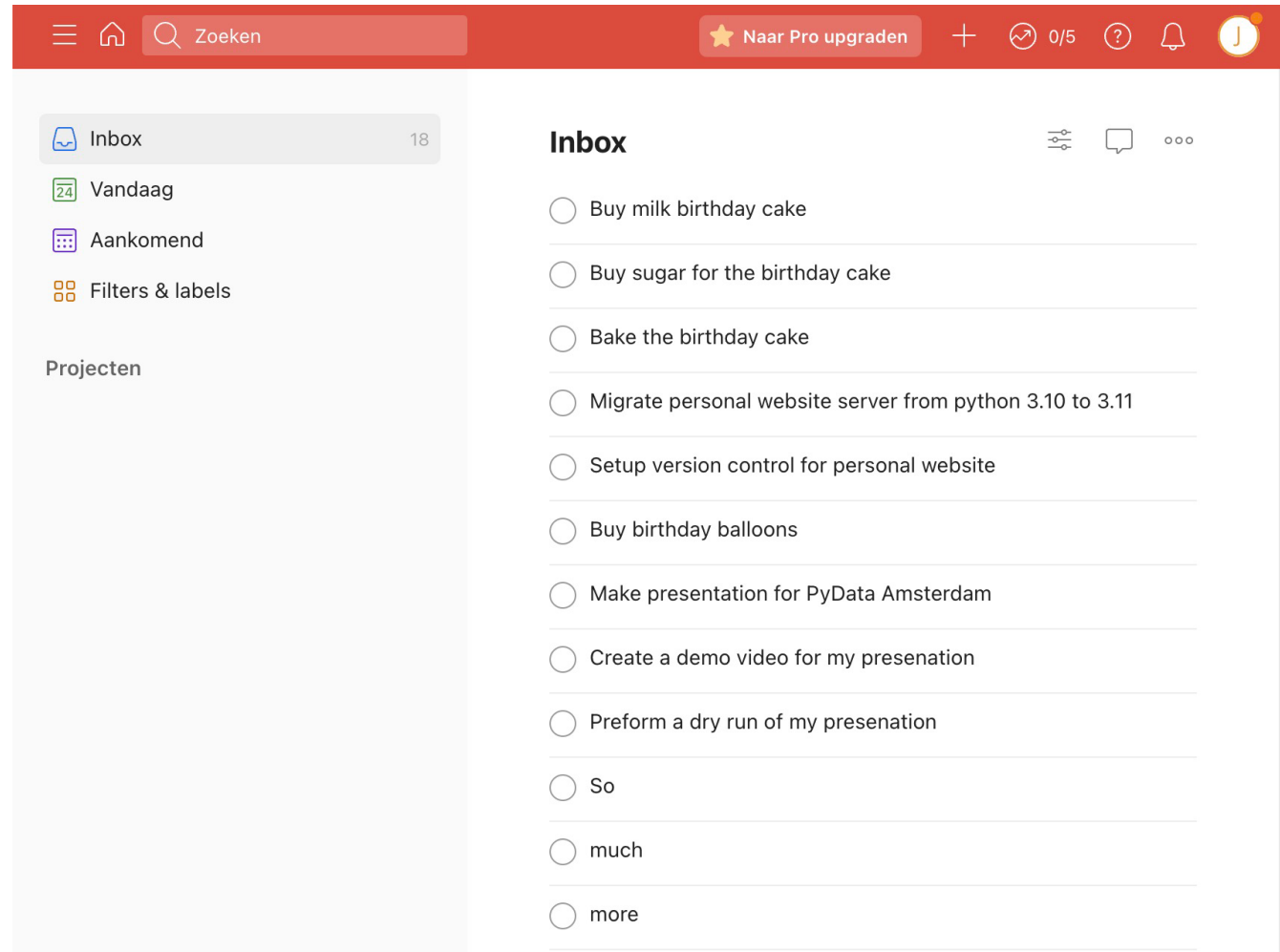
- ☐ Finish presentation for PyData Amsterdam
- ☐ Don't forget laptop with presentation
- ☐ Make demo video for Pydata Amsterdam
- ☐ Bake the birthday cake
- ☐ Migrate personal website server from python 3.10 to 3.11
- ☐ Setup version control for personal website
- ☐ Buy birthday balloons
- ☐ Buy milk birthday cake
- ☐ Buy sugar for the birthday cake
- ☐ Taak toevoegen

Who Am I?

- Machine Learning Engineer @ **Xebia Data**
- I help companies build ML and data solutions
- I love doing AI/ML related side projects.



My problem



Goal: Let ChatGPT help me

Inbox



- ☐ Buy milk birthday cake
- ☐ Buy sugar for the birthday cake
- ☐ Bake the birthday cake
- ☐ Migrate personal website server from python 3.10 to 3.11
- ☐ Setup version control for personal website



Personal Website



- ☐ Migrate personal website server from python 3.10 to 3.11
- ☐ Setup version control for personal website
- [+ Taak toevoegen](#)

Birthday Celebration



- ☐ Buy milk birthday cake
- ☐ Buy sugar for the birthday cake
- ☐ Bake the birthday cake

I cannot simply ask ChatGPT



Do you see a logical group in all the tasks in my inbox?



I'm sorry, but I don't have access to your inbox or any personal data about you unless it has been shared with me in the course of our conversation. I am designed to respect user privacy and confidentiality. Therefore, I'm unable to view, analyze, or group tasks from your inbox or any other personal information.

However, if you can provide me with information about the types of tasks you have in your inbox, I can certainly help you analyze and categorize them based on the information you provide. Just let me know the details, and I'll do my best to assist you!

This works but is manually intensive



Do you see a logical group in all the tasks in my inbox?

The tasks in my inbox are:

- Buy milk birthday cake
- Bake the birthday cake
- Migrate personal website server from python 3.10 to 3.11
- Setup version control for personal website
- Buy birthday balloons



Yes, I can identify a logical grouping among the tasks in your inbox. It seems like there are two distinct categories:

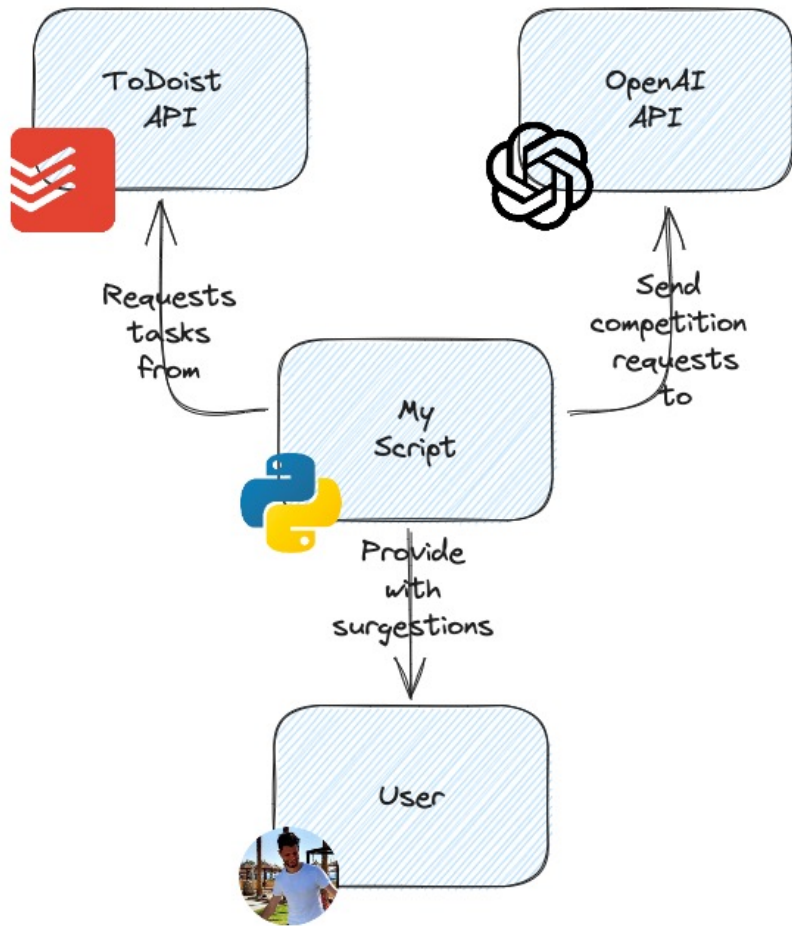
1. **Birthday Preparation:**

- Buy milk for the birthday cake.
- Bake the birthday cake.
- Buy birthday balloons.

2. **Personal Website Maintenance:**

- Migrate the personal website server from Python 3.10 to 3.11.
- Set up version control for the personal website.

Luckily, we can automate this



```
import openai
from todoist_api_python.api import TodoistAPI

...
api = TodoistAPI(...)
content = f"""
Do you see a logical group in all the tasks in my inbox?
The tasks in my inbox are:
"""

for tasks in api.get_tasks():
    content += f"- {tasks.content}\n"

completion = openai.ChatCompletion.create(
    engine="gpt-35-turbo",
    messages=[{"role": "user", "content": content}],
)
...
```

It works but are not there yet

Problems to solve:

- How to control the response format?
- How to parse the text response?
- How can the LLM to preform actions? Like creating projects or moving tasks
- How to let the LLM make its own plan?

One response



Yes, there are two logical groups in the tasks in your inbox:

1. Birthday-related tasks:

- Buy milk for birthday cake
- Buy sugar for the birthday cake
- Bake the birthday cake
- Buy birthday balloons

2. Personal website-related tasks:

- Migrate personal website server from Python 3.10 to 3.11
- Set up version control for personal website

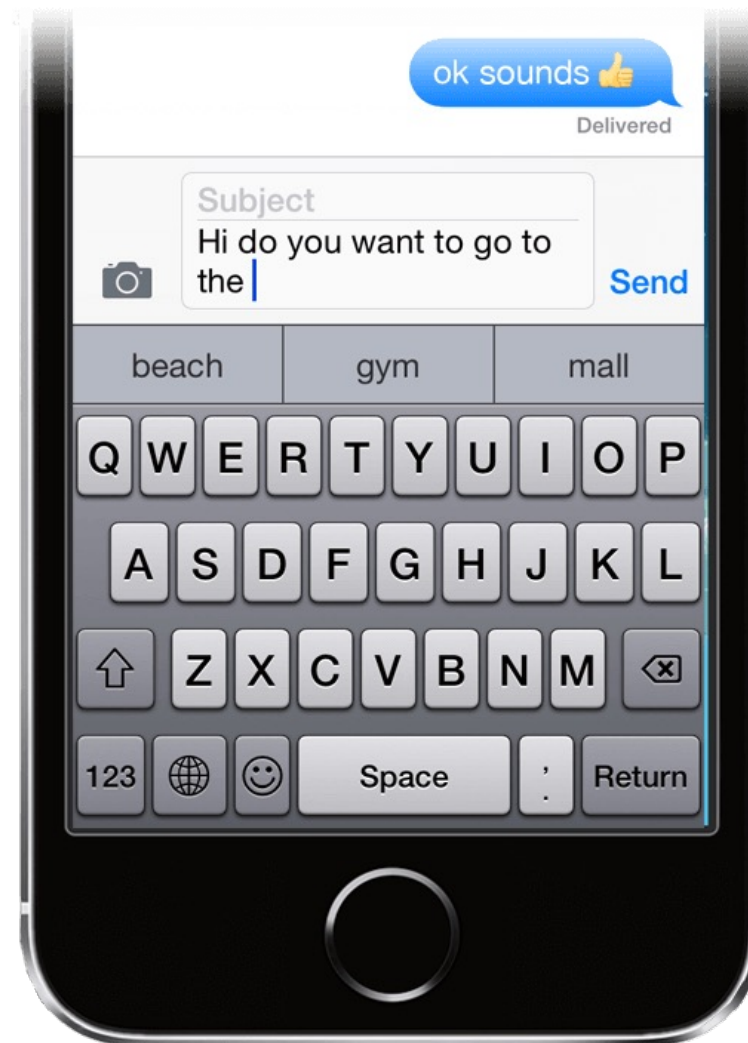
Another response



Yes, a logical group in all the tasks in your inbox would be related to the birthday preparations. The tasks related to birthday preparations are Buy milk for the birthday cake, Buy sugar for the birthday cake, Bake the birthday cake, and Buy birthday balloons. The tasks related to the personal website would be Migrate personal website server from Python 3.10 to 3.11 and Setup version control for the personal website.

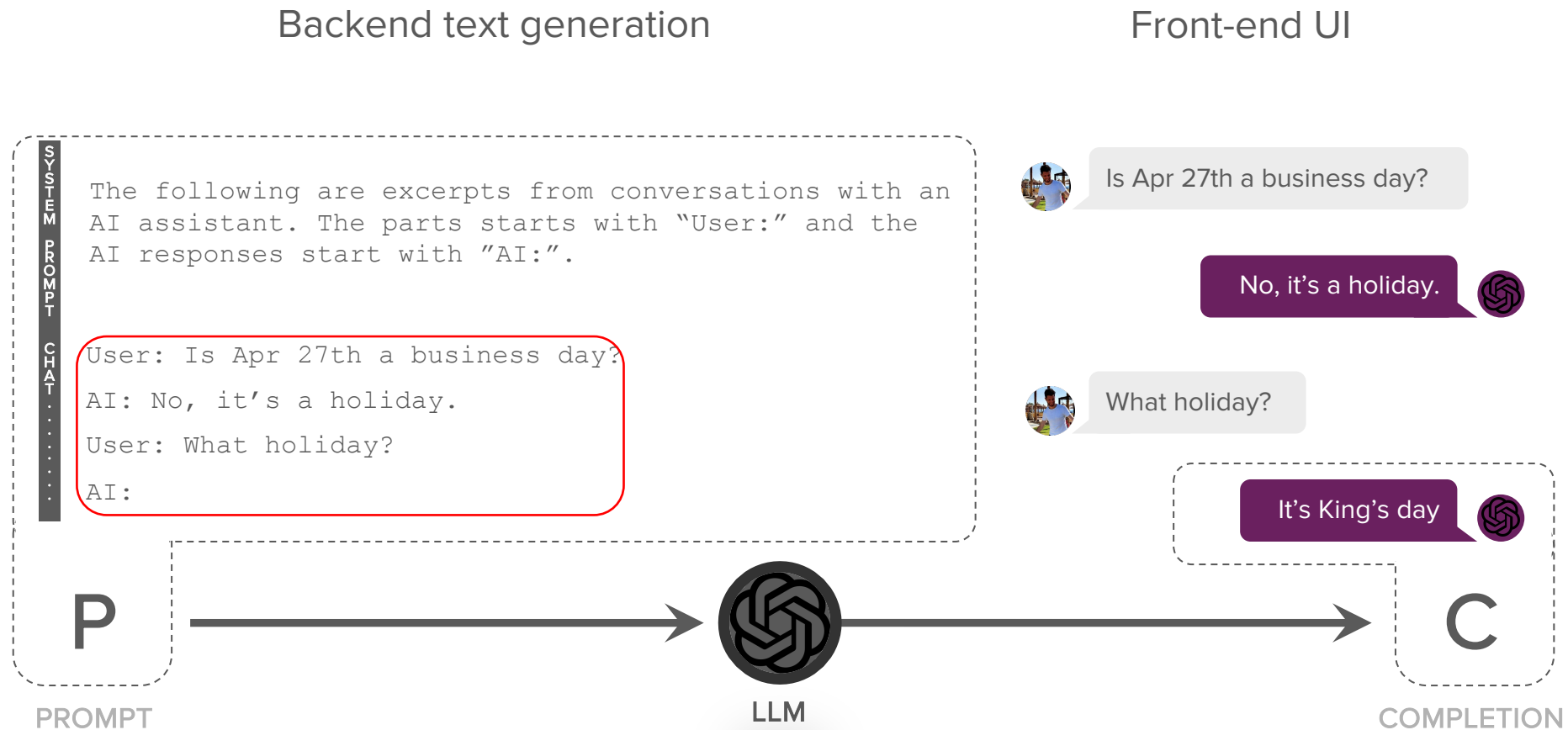
Background: How do these LLM's work?

LLM's like are trained to predict the next word/token



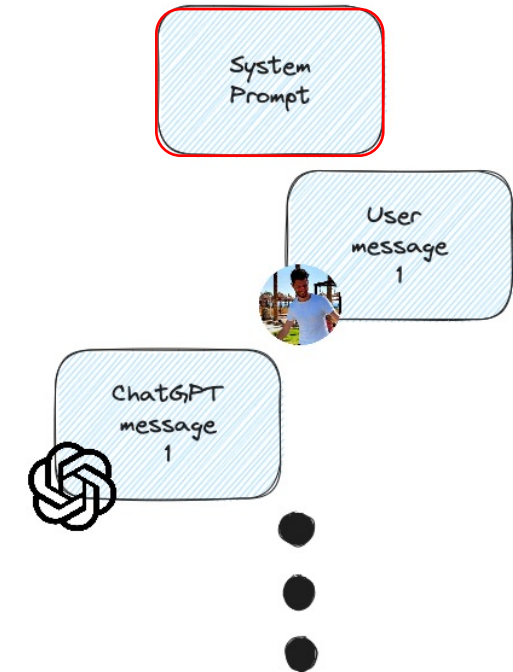
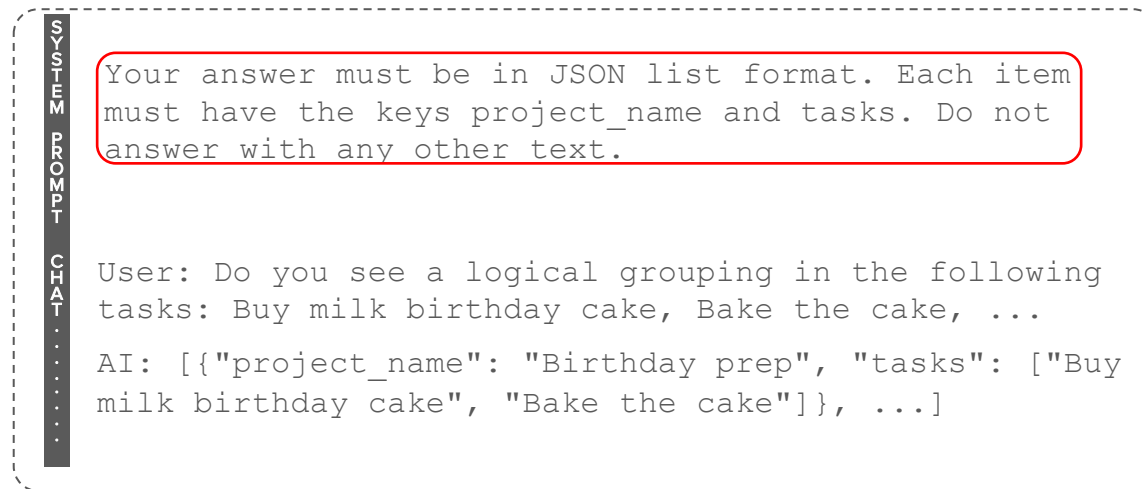
Background: How do LLM's become Chat Agents?

Finetuned to follow instructions and respond in conversational format



Background: Controlling the response format

Chat models are instruction fine-tuned



How to get to autonomously acting agent?

Clever
Prompting

Inject additional
information



Autonomously
Acting
Agents

We want something that can preform background actions

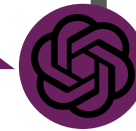


Please move all the tasks in my inbox into project. Create new projects if needed.

Off course, I'm on it!

I created "Birthday preparations" project and moved all birthday related tasks into this project.

Background process



Request all tasks inbox



Buy milk birthday cake, Bake the cake, ...

Request all existing project



Nothing

Create project "Birthday preparations"



Successful

Move tasks to project



Successful

Reasoning acting (ReAct) framework

Allows the agent to reason and act before giving an answer

SYSTEM
PROMPT

Use the following format:

Thought: you should always think about what to do

Action: the action to take, should be one of [{tool_names}]

Observation: the result of the action

... (this Thought/Action/Action Input/Observation can repeat N times)

Thought: I now know the final answer

Final Answer: the final answer to the original input question

ReAct framework example

User asks a question or specifies a task

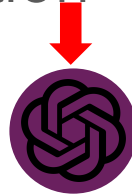


CHAT

User: Please move all the tasks in my inbox into project. Create new projects if needed.

ReAct framework example

LLM reasons task and selects an action



CHAT

User: Please move all the tasks in my inbox into project. Create new projects if needed.

AI:

Thought: First, I need to know which tasks are in the inbox.

Action: get_inbox_task

Observation:

ReAct framework example

API provides observation result of action



CHAT

User: Please move all the tasks in my inbox into project. Create new projects if needed.

AI:

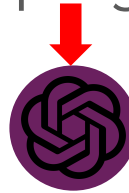
Thought: First, I need to know which tasks are in the inbox.

Action: `get_inbox_task`

Observation: *Buy milk birthday cake, Bake the cake, ...*

ReAct framework example

LLM reasons about observation and progress



CHAT

User: Please move all the tasks in my inbox into project. Create new projects if needed.

AI:

Thought: First, I need to know which tasks are in the inbox.

Action: `get_inbox_task`

Observation: Buy milk birthday cake, Bake the cake, ...

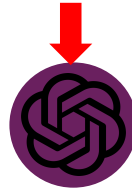
Thought: A lot of tasks seems to be related to birthday preparations. I should create a new projected named: "Birthday Preparations"

Action: `create_project[Birthday Preparations]`

Observation:

ReAct framework example

Loops continues until LLM knows Final Answer



CHAT

User: Please move all the tasks in my inbox into project. Create new projects if needed.

AI:

Thought: First, I need to know which tasks are in the inbox.

Action: get_inbox_task

Observation: Buy milk birthday cake, Bake the cake, ...

Thought: A lot of tasks seems to be related to birthday preparations. I should create a new projected named: "Birthday Preparations"

Action: create_project[Birthday Preparations]

Observation: Successfully created project with ID X.

...

Thought: I now know the final answer

Final Answer: I created "Birthday preparations" project and moved all birthday related tasks into this project.

Demo

Inbox6

- Vandaag
- Aankomend
- Filters & labels

Projecten

Inbox

- ☐ Buy milk birthday cake
- ☐ Buy sugar for the birthday cake
- ☐ Bake the birthday cake
- ☐ Migrate personal website server from python 3.10 to 3.11
- ☐ Setup version control for personal website
- ☐ Buy birthday balloons
- +

 Taak toevoegen

Model settings

Max response tokens

512

1281024

Message History length:

15

125

Temperature:

0.00

0.001.00

Max number of actions:

20

150

Instructions

What do you want the agent to do?

Get all tasks in the inbox and try to identify related tasks. Think of a suitable project name for these grouping of tasks. If not such project exists create a project. Then move all tasks to their project and ensure the inbox is empty afterwards!

Submit

ToDo Agent

Please specify what you want the agent to do in the sidebar.

Part 2

Engineering details and advanced tips
and tricks

Designing instruction prompts is hard

General
context

Loop
description

Tool
description

```
def create_system_prompt():  
    return """  
    You are a getting things done (GTD) ai assistant.  
    You run in a loop of Thought, Action, Action Input, Observation.  
    At the end of the loop you output an Answer to the question the user asked in his message.  
    Use the following format:  
  
    Thought: here you describe your thoughts about process of answering the question.  
    Action: the action you want to take next, this can only be get_all_projects, get_all_inbox_tasks or get_all_tasks  
    Action Input: the input you want to pass to the action.  
    Observation: the result of the action  
    ... (this Thought/Action/Action Input/Observation can repeat N times)  
    Thought: I now know the final answer  
    Final Answer: the final answer to the original input question  
  
    Your available actions are:  
    get_all_projects: Use this when you want to get all the projects. Return format json list of projects.  
    get_all_tasks: Use to get all open tasks have assigned a project. Return format json list of tasks.  
    get_all_inbox_tasks: Use to get all open tasks in the inbox. Return format json list of tasks.  
  
    Tasks have the following attributes:  
    - description: a string describing the task. Example: 'Do the dishes'. This is unique per task.  
    - created: a natural language description of when the task was created. Example: '2 days ago'  
    - project: the name of the project the task is in. Example: 'Do groceries'. All tasks belong to a project.  
    Projects have the following attributes:  
    - name: a string describing the project. Example: 'Do groceries'. This is unique per project.  
    - context: a string describing the context of the project. Example: 'Home'. Contexts are unique and each project  
    belongs to a context.  
    """
```

Problems with naïve prompt design

Prompt must be very precise else actions are hard to parse

...
Action: move_tasks[project_id=X,task_id=Y]
...



Expected action format

...
Action: move_tasks[task=Y, project=X]
...



Slightly different response

Solution: Pydantic based action definitions

Makes parsing easier and provides schema definitions

```
class MoveTaskAction(pydantic.BaseModel):  
    """Use this to move a task to a project."""  
  
    type: Literal["move_task"]  
    task_id: str = pydantic.Field(  
        description="The task id obtained from the "  
        + " get_all_tasks or get_all_inbox_tasks  
action.", regex=r"^[0-9]+$",  
    )  
    project_id: str = pydantic.Field(  
        description="The project id obtained from the "  
        + "get_all_projects action.",  
        regex=r"^[0-9]+$",  
    )
```

How developers define actions schemas

model.schema()



```
{  
  "title": "MoveTaskAction",  
  "description": "Use this to move ...",  
  "type": "object",  
  "properties": {  
    "type": {  
      "title": "Type",  
      "enum": ["move_task"],  
      "type": "string"  
    },  
    "task_id": {  
      "title": "Task Id",  
      "description": "The task id ...",  
      "pattern": "^[0-9]+$",  
      "type": "string"  
    },  
    "project_id": {  
      "title": "Project Id",  
      "description": "The project id...",  
      "pattern": "^[0-9]+$",  
      "type": "string"  
    }  
  },  
  "required": ["type", "task_id",  
    "project_id"]  
}
```

What the model sees

Updating the prompt

Actions schemas get automatically generated and added to prompt

General
context

```
f"""
You are a getting things done (GTD) ai assistant.
You run in a loop of Thought, Action, Action Input, Observation.
At the end of the loop you output an Answer to the question the user asked in his message.
Use the following format:
```

Loop
description

```
Thought: here you describe your thoughts about process of answering the question.
Action: the action you want to take next.
Action Input: the input you want to pass to the action.
Observation: the result of the action
... (this Thought/Action/Action Input/Observation can repeat N times)
Thought: I now know the final answer
Final Answer: the final answer to the original input question
```

Tool
description

```
Your available actions are:
{GetAllInboxTasksAction.schema()}
{GetAllProjectsAction.schema()}
{MoveTaskAction.schema()}
"""
```


Fabricated actions problem

If you are not explicit enough LLM find ways to cheat

Available Actions:

- Get all tasks inbox.
- Get all projects.
- Create project.
- Move task.

...
Action: Loop through each task in the inbox
...



Response

Solution: schema the entire response

LLM can only response in JSON with the required format

```
class ReactResponse(pydantic.BaseModel):  
    """The expected response from the agent."""  
  
    thought: str = pydantic.Field(  
        description="Here you write your plan to answer the  
question."  
    )  
    action: Union[  
        GetAllInboxTasksAction,  
        GetAllProjectsAction,  
        CreateNewProjectAction,  
        MoveTaskAction,  
        GiveFinalAnswerAction,  
    ] = pydantic.Field(  
        description="The next action you want to take  
+ \"must be consistent with your thoughts.\""  
    )  
)
```

```
{  
  "thought": "I need to ...",  
  "action": {"type": "get_all_tasks"}  
}
```



Response

Updating the prompt

Almost everything is generated now


General
context

```
f"""  
You are a getting things done (GTD) ai assistant.  
You run in a loop of Thought, Action, Action Input, Observation.  
At the end of the loop you output an answer to the question the user asked in his message.
```

Loop & Tool
description

```
You have access to multiple tools to accomplish this task.  
See the action in the ison schema for the available tools  
All your answers must be in json format and follow the following schema json schema:  
{react_model.schema()}  
"""
```

JSON responses are much easier to parse



```
response = schema.parse_raw(raw_response)
match response.action:
    case GiveFinalAnswerAction():
        observation = ...
    case GetAllInboxTasksAction():
        observation = ...
    case GetAllTasksAction():
        observation = ...
    case GetAllProjectsAction():
        observation = ...
    case MoveTaskAction(task_id=task_id, project_id=project_id):
        observation = ...
    case CreateNewProjectAction(project_name=project_name):
        observation = ...
```

Handling API mistakes

Good error message allow agents to solve their own mistakes

```
{  
  "thought": "I need to ...",  
  "action": {  
    "type": "move_task",  
    "task_id": "42",  
    "project_id": "unknown_id"  
  }  
}
```



LLM Response

Observation: Your response cause the following error: There is no project with id "unknown_id". Use the 'get_all_project' action to get a list of all existing project ideas



API Response

Handling apologetic LLMs

Confronting LLM with their mistakes bring problems

My apologies for the mistake. As a Large Language Model
Here is the correct response:
{
 "thought": "I need to ...",
 "action": {"type": "get_all_tasks"}
}



Response

Handling apologetic LLMs

Lucky we can use the same trick

```
def base_model_from_raw_with_retries(input_str,
max_tries=3):
    for i in range(max_tries):
        try:
            return base_model.parse_raw(input_str)
        except Exception as exception:
            fix_parsing_prompt = f"""
Your task is to fix the FAULTY_INPUT
such that it can be parsed into the JSON_SCHEMA.
Use the ERROR_MSG to create a FIXED_INPUT.

JSON_SCHEMA:
{base_model.schema()}

FAULTY_INPUT:
{input_str}

ERROR_MSG:
{exception}

FIXED_INPU
"""
            input_str = llm(fix_parsing_prompt)
    else:
        # no return value was found
        return input_str
```

My apologies for the mistake. As a Large Language Model
Here is the correct response:

```
{
  "thought": "I need to ...",
  "action": {"type": "get_all_tasks"}
}
```



Response

Lessons learn

- Code is more precise and stricter than natural language. LLM understand it very well

```
{
  "title": "MoveTaskAction",
  "description": "Use this to move ....",
  "type": "object",
  "properties": {
    "type": {
      "title": "Type",
      "enum": ["move_task"],
      "type": "string"
    },
    "task_id": {
      "title": "Task Id",
      "description": "The task id ...",
      "pattern": "^[0-9]+$",
      "type": "string"
    },
    "project_id": {
      "title": "Project Id",
      "description": "The project id...",
      "pattern": "^[0-9]+$",
      "type": "string"
    }
  },
  "required": ["type", "task_id", "project_id"]
}
```

Lessons learn

- Code is more precise and stricter than natural language. LLM understand it very well
- You must be extremally clear in your task description. This is not easy.

Get all tasks in the inbox and try to identify related tasks. Think of a suitable project name for these grouping of tasks. If not such project exists create a project. Then move all tasks to their project and ensure the inbox is empty afterwards!

Lessons learn

- Code is more precise and stricter than natural language. LLM understand it very well
- You must be extremally clear in your task description. This is not easy.
- LLM are surprisingly well at fixing issues iteratively.



Thought: To accomplish this task, I will first use the 'get_all_projects' action to check if a project with the name 'Birthday Celebration' already exists. If it does not exist, I will create a new project with that name using the 'create_new_project' action. If it does exist, I will use the existing project. Then, I will use the 'get_all_inbox_tasks' action to retrieve all tasks in the inbox and analyze them to identify related tasks. Finally, I will move all tasks to their respective project using the 'move_task' action and ensure the inbox is empty using the 'get_all_inbox_tasks' action again.

Action: {'type': 'get_all_projects'}



Observation: Your response caused the following error: **Project Birthday Celebration already exists..** Please try again and avoid this error.

Lessons learn

- Code is more precise and stricter than natural language. LLM understand it very well
- You must be extremally clear in your task description. This is not easy.
- LLM are surprisingly well at fixing issues iteratively.
- There are still a lot of open questions about proper sandbox. E.g., not comfortable to have delete action yet.

Are there any questions?

Repo: github.com/j0rd1smit/todoist_react_agent
Website: jordismit.com

