Documentação da API da plataforma FutureHouse

Documentação e tutoriais para futurehouse-client, um cliente para interação com endpoints da plataforma FutureHouse.

Instalação

```
uv pip install futurehouse-client
```

Início rápido

```
from futurehouse_client import FutureHouseClient, JobNames
from pathlib import Path
from aviary.core import DummyEnv
import ldp

client = FutureHouseClient(
    api_key="your_api_key",
)

task_data = {
    "name": JobNames.CROW,
    "query": "Which neglected diseases had a treatment developed by
artificial intelligence?",
}

task_response = client.run_tasks_until_done(task_data)
```

Um exemplo de início rápido pode ser encontrado no arquivo client_notebook.ipynb ↗ , onde mostramos como enviar e recuperar uma tarefa, passar a configuração de tempo de execução para o agente e fazer perguntas de acompanhamento sobre a tarefa anterior.

Funcionalidades

O cliente FutureHouse implementa um RestClient (chamado `FutureHouseClient`) com as seguintes funcionalidades:

- Tarefa simples em execução : `run_tasks_until_done(TaskRequest)` ou `await arun_tasks_until_done(TaskRequest)`
- Tarefas assíncronas : `get_task(task_id)` ou `aget_task(task_id)` e `create_task(TaskRequest)` ou `acreate_task(TaskRequest)`

Para criar um `FutureHouseClient`, você precisa passar uma chave de API da plataforma FutureHouse (veja Autenticação):

```
from futurehouse_client import FutureHouseClient

client = FutureHouseClient(
    api_key="your_api_key",
)
```

Autenticação

Para usar o `FutureHouseClient`, você precisa se autenticar. A autenticação é feita fornecendo uma chave de API, que pode ser obtida diretamente na sua página de perfil na plataforma FutureHouse ↗.

Tarefa simples em execução

Na plataforma FutureHouse, definimos a combinação implantada de um agente e um ambiente como um `job`. Para invocar uma tarefa, precisamos enviar um `task` (também chamado de `query`) para ela. Este método `FutureHouseClient` pode ser usado para enviar tarefas/consultas para tarefas disponíveis na plataforma FutureHouse. Usando uma `FutureHouseClient` instância, você pode enviar tarefas para a plataforma chamando o `create_task` método , que recebe um `TaskRequest` (ou um dicionário com `kwargs`) e retorna o ID da tarefa. Com o objetivo de simplificar ao máximo o envio de tarefas, criamos um `JobNames` `enum` que contém os tipos de tarefas disponíveis.

Os empregos suportados disponíveis são:

Pseudônimo	Nome do trabalho	Tipo de tarefa	Descrição
`JobNames.CROW`	`job-futurehouse-paperqa2`	Pesquisa rápida	Faça uma pergunta sobre fontes de dados científicos e receba uma resposta citada e de alta precisão. Desenvolvido com PaperQA2 ↗.
`JobNames.FALCON`	`job-futurehouse-paperqa2-deep`	Busca profunda	Utilize uma infinidade de fontes para pesquisar profundamente. Receba um relatório detalhado e estruturado como resposta.

`JobNames.OWL`	`job-futurehouse-hasanyone`	Pesquisa de precedentes	Anteriormente conhecido como HasAnyone, pergunte se alguém já fez algo na área científica.
`JobNames.PHOENIX`	`job-futurehouse-phoenix`	Tarefas de Química	Uma nova iteração do ChemCrow, o Phoenix usa ferramentas de quimioinformática para fazer química. Ótimo para planejar sínteses e projetar novas moléculas.
`JobNames.DUMMY`	`job-futurehouse-dummy`	Tarefa fictícia	Esta é uma tarefa fictícia, principalmente para fins de teste.

Usando `JobNames`, o envio da tarefa fica assim:

```
from futurehouse_client import FutureHouseClient, JobNames

client = FutureHouseClient(
    api_key="your_api_key",
)

task_data = {
    "name": JobNames.OWL,
    "query": "Has anyone tested therapeutic exerkines in humans or
NHPs?",
}

task_response = client.run_tasks_until_done(task_data)

print(task_response.answer)
```

Ou se estiver executando código assíncrono:

```
import asyncio
from futurehouse_client import FutureHouseClient, JobNames


async def main():
    client = FutureHouseClient(
        api_key="your_api_key",
    )

    task_data = {
        "name": JobNames.OWL,
        "query": "Has anyone tested therapeutic exerkines in humans or
NHPs?",
    }

    task_response = await client.arun_tasks_until_done(task_data)
    print(task_response.answer)
    return task_id


# For Python 3.7+
if __name__ == "__main__":
    task_id = asyncio.run(main())
```

Observe que, tanto no código síncrono quanto no async, coleções de tarefas podem ser fornecidas ao cliente para execução em lote:

```
import asyncio
from futurehouse_client import FutureHouseClient, JobNames


async def main():
    client = FutureHouseClient(
        api_key="your_api_key",
    )

    task_data = [{
        "name": JobNames.OWL,
        "query": "Has anyone tested therapeutic exerkines in humans or
NHPs?",
    },
    {
        "name": JobNames.CROW,
        "query": "Are there any clinically validated therapeutic
exerkines for humans?",
    }
    ]

    task_responses = await client.arun_tasks_until_done(task_data)
    print(task_responses[0].answer)
    print(task_responses[1].answer)
    return task_id


# For Python 3.7+
if __name__ == "__main__":
    task_id = asyncio.run(main())
```

`TaskRequest` também pode ser usado para enviar trabalhos e possui os seguintes campos:

Campo	Tipo	Descrição
eu ia	UUID	Identificador de trabalho opcional. Um UUID será gerado se não for fornecido
nome	str	Nome do trabalho a ser executado, por exemplo `job-futurehouse-paperqa2`, ou usando o `JobNames` para conveniência: `JobNames.CROW`

consulta	str	Consulta ou tarefa a ser executada pelo trabalho
configuração_de_tempo_de_execução	Configuração de tempo de execução	Parâmetros de tempo de execução opcionais para o trabalho

`runtime_config` pode receber um `AgentConfig` objeto com os kwargs desejados. Verifique os `AgentConfig` campos disponíveis na documentação do LDP ↗ . Além do `AgentConfig` objeto, também podemos passar `timeout` e `max_steps` para limitar o tempo de execução e o número de passos que o agente pode realizar.

```
from futurehouse_client import FutureHouseClient, JobNames
from futurehouse_client.models.app import TaskRequest

client = FutureHouseClient(
    api_key="your_api_key",
)

task_response = client.run_tasks_until_done(
    TaskRequest(
        name=JobNames.OWL,
        query="Has anyone tested therapeutic exerkines in humans or
NHPs?",
    )
)

print(task_response.answer)
```

Um valor `TaskResponse` será retornado ao usar nossos agentes. Para Coruja, Corvo e Falcão, usamos como padrão uma subclasse, `PQATaskResponse` que possui alguns atributos-chave:

Campo	Tipo	Descrição
responder	str	Resposta à sua pergunta.
resposta_formatada	str	Resposta especialmente formatada com referências.

Se usar essa `verbose` configuração, muito mais dados poderão ser extraídos do seu `TaskResponse`, que existirão em todos os agentes (não apenas Owl, Crow e Falcon).

```
from futurehouse_client import FutureHouseClient, JobNames
from futurehouse_client.models.app import TaskRequest

client = FutureHouseClient(
    api_key="your_api_key",
)

task_response = client.run_tasks_until_done(
    TaskRequest(
        name=JobNames.OWL,
        query="Has anyone tested therapeutic exerkines in humans or
NHPs?",
    ),
    verbose=True,
)

print(task_response.environment_frame)
```

Nesse caso, a `TaskResponseVerbose` terá os seguintes campos:

| Campo | Tipo | Descrição | | ---------- | ---- | ---------- | --- | | agent_state | dict | Objeto grande com todos os estados do agente durante o andamento da sua tarefa. | | environment_frame | dict | Objeto grande aninhado com todos os dados do ambiente. Para ambientes PQA, inclui contextos, metadados de papel e respostas. | | metadata | dict | Metadados extras sobre sua consulta. | |

Continuação da tarefa

Após o envio de uma tarefa e a resposta, a plataforma FutureHouse permite que você faça perguntas complementares à tarefa anterior. Isso também é possível por meio da API da plataforma. Para isso, podemos usar o `runtime_config` método discutido na seção Execução de tarefa simples .

```
from futurehouse_client import FutureHouseClient, JobNames

client = FutureHouseClient(
    api_key="your_api_key",
)

task_data = {"name": JobNames.CROW, "query": "How many species of birds
are there?"}

task_id = client.create_task(task_data)

continued_task_data = {
    "name": JobNames.CROW,
    "query": "From the previous answer, specifically,how many species
of crows are there?",
    "runtime_config": {"continued_task_id": task_id},
}

task_result = client.run_tasks_until_done(continued_task_data)
```

Tarefas assíncronas

Às vezes, você pode querer enviar várias tarefas enquanto consulta os resultados posteriormente. Dessa forma, você pode fazer outras coisas enquanto aguarda uma resposta. A API da plataforma também oferece suporte para isso, em vez de esperar por um resultado.

```
from futurehouse_client import FutureHouseClient

client = FutureHouseClient(
    api_key="your_api_key",
)

task_data = {"name": JobNames.CROW, "query": "How many species of birds
are there?"}

task_id = client.create_task(task_data)

# move on to do other things

task_status = client.get_task(task_id)
```

`task_status` contém informações sobre a tarefa. Por exemplo, seus campos `status`, `task`, `environment_name` e `agent_name`, e outros campos específicos da tarefa. Você pode consultar o status continuamente até que esteja `success` pronto antes de prosseguir.