AI-Engine Python SDk

Introduction

AI Engine SDK provides an easy-to-use interface for integrating advanced AI functionalities into your applications. With a simple API and comprehensive documentation, developers can effortlessly leverage powerful AI tools and enhance their projects.

Getting Started

Prerequisites

Before you begin, ensure you have met the following requirements:

- Python 3.10 or higher
- installed on your system.
- A validAgentverse API key
- . You can obtain this fromAgentverse(opens in a new tab)
- .

i To find out how to generate an API KEY check out the documentation regardingAgentverse API keys(opens in a new tab) .

Installation

You can install the AI Engine SDK using either Poetry or pip. Follow the commands below.

poetry

add

ai-engine-sdk pip

install

ai-engine-sdk **i** Explore the additional resources we have on AI Engine compatible Agents:

- Make your agents AI Engine compatible
- Agent Functions
- Register Agent Functions
- DeltaV

SDK overview

Creating the AI Engine client object

To set up and initialize the AI Engine client with your API key, use the following code snippet. This client will allow you to interact with the AI Engine API, enabling you to start sessions, get function groups, and manage messages.

from ai_engine_sdk import AiEngine ai_engine : AiEngine =

AiEngine (api_key)

Querying Function Groups

The AI Engine SDK allows you to query different groups of functions, such asprivate functions ,public functions , orFetch.ai-verified functions . You can filter function groups using the respective methods provided by the SDK based on your needs.

Querying Private Functions

Private functions are those that you have created and are not accessible to the public or other users. To access your private functions, use theget_private_function_groups() method. This will return all functions under your account, giving you access to personal or proprietary models and algorithms.

Example

private

await ai_engine . get_private_function_groups ()

private_functions

next ((g for g in private if g.name ==

"My Functions"), None)

Querying Public Functions

Public functions are accessible by anyone on the platform. These functions include your publicly available functions and functions verified by Fetch.ai. To access public function groups, use theget_public_function_groups() method. You can further filter by two important groups:Public andFetch Verified .

public

await ai_engine . get_public_function_groups ()

Access your public functions

public_functions

next ((g for g in public if g.name ==

"Public"), None)

Access Fetch.ai-verified functions

fetch_verified_group

next ((g for g in public if g.name ==

"Fetch Verified"), None)

Sharing function groups

Allow other users to use functions, under a function-group, without replicating that function or allowing them alter those functions or function-group data.

If you wish to give access to a certain function-group (use, not alter the data/state of it), you can use the following method in the following way:

assuming ai_engine is a valid instance of AiEngine

what_function_group_identifier_i_want_share

"normally-this-is-an-uuid4" user_email_i_want_to_share_the_function_group_with =

"random@domain.com" await ai_engine . share_function_group (function_group_id = what_function_group_identifier_i_want_share, target_user_email = user_email_i_want_to_share_the_function_group_with) Now, if you were requesting the available function-groups for the user with email assigned to thetarget_user_email argument, the function-group with the id assigned tofunction_group_id .

You can check that by using theAiEngine.get_function_groups method.

Creating a session with the AI Engine.

This code starts a session with the AI Engine using the UUID of the selected function group.

session

await ai_engine . create_session (function_group = public_group.uuid)

Starting the conversation with an arbitrary objective

await session . start (objective)

Querying new messages

This line asynchronously queries the AI engine to retrieve a list of messages related to the current session. The messages are returned as a list ofApiBaseMessage objects, which can include various types of messages such as agent messages, AI engine messages, task selections, confirmations, and session stop messages. These messages are then processed to determine the next actions in the session workflow.

while

True : messages : list [ApiBaseMessage]

=

await session . get_messages () sleep (4)

Handling Different Types of Messages

Task Selection Message (is_task_selection_message

)

This message is generated when the AI engine suggests functions based on the initial objective or provides multiple options for a function.

Agent Message (is_agent_message

)

This is a regular question from the AI Engine that the user needs to reply to with a string.

AI Engine Message (is_ai_engine_message

)

This message type doesn't require a user response; it simply notifies the user about something.

Confirmation Message (is_confirmation_message

)

This message is sent when the AI Engine has gathered all necessary inputs for the agent's function, indicating that the context is complete.

Stop Message (is_stop_message

)

This message is sent when the session has ended, and the AI Engine no longer expects any replies from the user.

i All message types (expect for the AI engine message and stop message) expects a response from the user.

SDK Methods for Replying

Task Selection Message

Usesession.submit_task_selection .

Agent Message

Usesession.submit_response .

Confirmation Message

Use eithersession.submit_confirmation to confirm, orsession.reject_confirmation to reject the context generated by the AI engine.

Deleting session

After finishing a conversation with the AI Engine, you can delete the session by using the following command.

await session . delete ()

Example Usage of the SDK

The following example demonstrates how to use the AI Engine SDK to interact with the AI Engine. The script sets up the client, queries function groups, creates a session, and handles different types of messages in a loop.

Self hosted run_example.py import asyncio import logging import os import sys from time import sleep from ai_engine_sdk import (AiEngine , is_agent_message , is_ai_engine_message , is_confirmation_message , is_stop_message , is_task_selection_message , TaskSelectionMessage) from ai_engine_sdk import ApiBaseMessage , FunctionGroup

logger

logging . getLogger (**name**)

api_key

os . getenv ("AV_API_KEY" , "") interaction_user_prompt_header =

f " \n\n Interaction time"

async

def

main (): logger . debug (" Starting example execution") ai_engine =

AiEngine (api_key)

function_groups : list [FunctionGroup]

=

await ai_engine . get_function_groups ()

public_group

next ((g for g in function_groups if g.name ==

"Fetch Verified"), None) if public_group is

None : raise

Exception ('Could not find "Public" function group.')

session

await ai_engine . create_session (function_group = public_group.uuid) default_objective :

str

=

"Find a flight to warsaw."

logger . info (interaction_user_prompt_header) objective =

input (f " \n What is your objective [default: { default_objective }]: ")

or default_objective await session . start (objective)

try : empty_count =

0 session_ended =

False

while empty_count <

100 : messages : list [ApiBaseMessage]

=

await session . get_messages () if

len (messages)

==

0 : empty_count +=

1 else : empty_count =

0

message : ApiBaseMessage for message in messages : if

is_task_selection_message (message_type = message.type): task_selection_message : TaskSelectionMessage = message

logger . info (interaction_user_prompt_header) print ("Please select a key from the list below:\n") for _ , option in task_selection_message . options . items (): print (f "➡ { option.key } -> { option.title } ") option_key =

str (input ("\nEnter task key: "))

check the index

if option_key not

in task_selection_message . options . keys (): raise

Exception (f " Invalid task number. \n You selected: { option_key } ") logger . debug (option_key) await session . submit_task_selection (message, [task_selection_message.options[option_key]]) del task_selection_message elif

is_agent_message (message): logger . info (interaction_user_prompt_header) print (message.text. capitalize ()) response =

input ("✍ (enter to skip): ") if response ==

"exit" : break

if response !=

"" : await session . submit_response (message, response) elif

is_ai_engine_message (message): logger . info (f " \n ℹ Informative message \n\n ---> ✨ { message.text } ") sleep (3.5) elif

is_confirmation_message (message_type = message.type): logger . info (interaction_user_prompt_header) print ("Confirm:" , message.payload) response =

input ("\nPress enter to confirm, otherwise explain issue:\n")

if response ==

"" : await session . submit_confirmation (message) else : await session . reject_confirmation (message, response) elif

is_stop_message (message):

logger . info ("\n Session has ended, thanks! ") session_ended =

True break

if the session has concluded then break

if session_ended : break

logger . info (f " \n Processing \n ") sleep (1.5) logger . debug (f "No messages: { empty_count } times in a row")

except

Exception

as e : logger . debug (f "Unhandled exception: { e } ") print ("Error" , e) raise e finally :

clean up the session

await session . delete ()

if

name

==

"**main**" : logging . basicConfig (stream = sys.stdout, level = logging.DEBUG,

level=logging.INFO,

format

' %(asctime)s

%(levelname)s

%(module)s : %(message)s ' , datefmt = "%H:%M:%S") asyncio . run (main ()) Last updated on October 17, 2024

Was this page helpful?

You can also leave detailed feedback[on Github]

On This Page