Interpreter API

The Interpreter API is currently in development. This REST API service will provide human-readable information regarding transactions, addresses and more.

How It Works

The Transactions Interpreter Service is designed to process blockchain transactions by extracting and analyzing relevant information to generate a summary of the transaction. The process follows several distinct phases:

1. Transaction Hash Reception
2. : The service receives a transaction hash, which triggers the processing pipeline.
3. Data Retrieval
4. : The service automatically retrieves necessary transaction data from the blockchain using the provided transaction hash.
5. Initial Validation
6. : The input data undergoes validation to ensure authenticity and correctness before further processing.
7. Data Processing
8. : The retrieved transaction data is processed, involving multiple phases:
9.
    1. Pre-LLM Phase
10.
    1. : During this phase, the service attempts to classify and interpret the on-chain actions using predefined algorithms. This step includes determining the type of action (e.g., transfer, minting, etc.) based on the transaction data.
11.
    1. LLM Phase
12.
    1. : If the type of action cannot be determined algorithmically in the Pre-LLM phase, the service leverages a Language Learning Model (LLM) to infer the type of on-chain action.
13. Transaction Summary Generation
14. : After the classification, the service generates a summary of the transaction, which is returned as the response. The summary includes key details about the transaction and is structured according to a specific template.

Response Structure

The response from the Transactions Interpreter Service is structured as follows (will vary depending on the specific case):

Success Response

```

Copy { "success":true, "data": { "summaries":[ { "summary_template":"{action_type} {amount} {token} to {to_address}",
"summary_template_variables":{ "action_type":{ "type":"string", "value":"Transfer" }, "amount":{ "type":"currency",
"value":"7800" }, "token":{ "type":"token", "value":{ "address":"0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48",
"circulating_market_cap":"34593435047.40006", "decimals":"6", "exchange_rate":"1.001", "holders":"2761518",
"icon_url":"https://assets.coingecko.com/coins/images/6319/small/usdc.png?1696506694", "name":"USD Coin",
"symbol":"USDC", "total_supply":"25611927821576420", "type":"ERC-20", "volume_24h":"8858197053.476923" } },
"to_address":{ "type":"address", "value":{ "hash":"0x242ba6d68FfEb4a098B591B32d370F973FF882B7" } } } } ],
"debug_data":{ "transaction_hash":"0xea7cbead745789346343435cf2a0910887ab0ebef08af07cbd2f0920a2c628da",
"model_classification_type":"transfer", "post_llm_classification_type":"transfer", "is_prompt_truncated":false,
"summary_template":{ "transfer":{ "template_name":"Generic-Transfer", "template_vars":{ "strippedInput":"",
"decodedInput":"", "tokenTransfers":[], "isErc20Transfer": true, "erc20Amount": "7800", "isNftTransfer": false, "token": {
"address":"0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48", "circulating_market_cap":"34593435047.40006",
"decimals":"6", "exchange_rate":"1.001", "holders":"2761518",
"icon_url":"https://assets.coingecko.com/coins/images/6319/small/usdc.png?1696506694", "name":"USD Coin",
"symbol":"USDC", "total_supply":"25611927821576420", "type":"ERC-20", "volume_24h":"8858197053.476923" },
"nftAmount": null, "toAddress": { "hash":"0x242ba6d68FfEb4a098B591B32d370F973FF882B7" } } } } } } }

```

Response Variable Types

The response from the service can contain the following types of data:

- number
- : Numerical values.
- string
- : Textual data.

- array
- : An array of items.
- currency
- : Represents a monetary value.
- token
- : Information related to a cryptocurrency or token.
- address
- : Blockchain address information.
- timestamp
- : A date and time representation.
- domain
- : Domain name data.
- code
- : Code snippets or encoded data.
-

These types are used to ensure that the data returned in the summary is consistent and properly structured.

Error Handling

If the service encounters an error, it returns a structured error response:

- Error Response
- :
-

```

Copy { "success":false, "error": { "id":"v_ap_rd_1", "code":"INVALID_API_PARAMS", "message":"Invalid API parameters", "error_data":[ { "parameter":"data", "message":"Invalid parameter data. Please ensure the input is well formed." } ] } }

```

Attributes of the Error Object

- id
- : A unique identifier for the error, used for debugging purposes.
- code
- : A code representing the type of error.
- message
- : A user-friendly message that can be displayed on the front end.
- error_data
- : An array containing details about the invalid parameters.
-

Last updated1 month ago