Galadriel Documentation

For developers
Quickstart
This quickstart covers two things:

Deploying an AI app to the Galadriel devnet
Calling your deployed AI app and viewing the results

Prerequisites
A Galadriel devnet account. We recommend creating a new EVM account for development purposes.
Some devnet tokens on the account you are using.
node and npm installed on your machine.

Deploying a contract on Galadriel devnet
1
devnet tokens

Get yourself some devnet tokens from the faucet.

2
Clone repository

Clone the repo that contains Galadriel example contracts and the oracle implementation.

```
git clone https://github.com/galadriel-ai/contracts
cd contracts/contracts
```

3
Setup environment

Use the template.env file to create a .env file:

```
# Starting in repo root
cp template.env .env
```

Then, modify the .env file:

Set PRIVATE_KEY_GALADRIEL to the private key of the account you want to use for deploying the contracts to Galadriel devnet.
Set ORACLE_ADDRESS to the current devnet oracle address provided by Galadriel team: 0x68EC9556830AD097D661Df2557FBCeC166a0A075.
4
Install dependencies

Install the dependencies using npm:

```
# In repo root -> /contracts
npm install
```

5
Deploy contract to devnet

The quickstart contract can be deployed with a simple script:

```
npm run deployQuickstart
```

The output of the script will show the deployed contract address. Store this and export it in terminal to call the contract later:

```
# Replace with your own contract address
```

```
export QUICKSTART_CONTRACT_ADDRESS=0x...
```

Calling your contract
Prerequisites:

You've deployed the quickstart contract in the previous step, and stored the contract address.
1
Execute the following command to run a script that calls the deployed contract:

```
npm run callQuickstart
```

The script will interactively ask for the input (DALL-E prompt: what image should be generated) and then call the contract with the input. The output, when ready, will be printed to the console.

If this step fails, make sure you have set the QUICKSTART_CONTRACT_ADDRESS environment variable to the address of your deployed contract.

You're done! You've successfully created and called your own on-chain AI app.

What's next?

Continue developing your own contracts. See the contracts readme and the Solidity reference for more.
Read how Galadriel works.
Explore use cases for inspiration and examples of full dApps including a frontend.

For developers
How it works

Goal
The goal of Galadriel is to enable on-chain autonomous AI apps, especially agents. Specifically, we want to enable developers to create on-chain programs which are capable of:

calling LLMs and multimodal models, including from closed-source API-providers,
permanently storing large amounts of data and retrieving relevant parts of it on-demand, including using embedding-based semantic search,
making requests to external services like search engines,
executing all the above in a loop running for several minutes.
Why these requirements? First, models are the core of any AI app. Second, storage allows extending and specializing the models' knowledge with relatively little effort. Third, external tools are crucial for extending the AI's capability to act in the real world, in real time. And fourth, long runtimes are a fact of life for state of the art large models, and looping a model (which further extends inference time) improves the AI's reasoning ability.

Design
The core of how Galadriel enables on-chain AI is the oracle.

The oracle enables contracts to make external API calls (including LLMs). It offers an interface for Solidity contracts on the Galadriel chain to call out LLMs or other models, use tools, and retrieve data.

The oracle is implemented as a contract that is called asynchronously and has an off-chain component. This is similar in architecture to ChainLink. However, because tool calls (e.g. web searches) and LLM calls do not produce deterministic results nor cannot be averaged, consensus on those requests is not possible. To solve this, we execute the oracle in a trusted execution environment — see the TEE section.

To make your own on-chain AI, you need to build your Solidity contract in a specific way to interact with the oracle. Making a call to most contracts on EVM chains is synchronous: the call is made, and the result is returned immediately. However, the oracle is asynchronous: the call is made, and the result is returned later, when the oracle has finished processing the request off-chain. The async nature of the oracle is because on-chain programs cannot typically execute long-running tasks (block time is the ceiling on execution time).

Due to the above, to make a call to an LLM, generative image model, external tool, or anything else via the oracle you need to use a callback function. This function is called by the oracle when the result is ready.

The oracle sits atop a parallel, EVM-compatible Layer 1, based on Cosmos SDK & Sei v2.

Usage patterns

Based on this, here are the typical usage patterns for Galadriel.

Simple generative image model call

Assume you want to make one call to a generative image model during the execution of your contract. This is what you need to make e.g. a generative AI NFT minting dApp.

To do so, you need to add two things to your contract:

Make a call to the generative image model. This is done by calling a function on the oracle.
Add a callback function to your contract. This function is called by the oracle when the result is ready.
And the execution flow is as follows (see the diagram below):

Your contract calls the oracle contract to make a call to the generative image model — say, DALL-E.
The oracle backend makes a call to DALL-E API, off-chain.
(About 10-20 seconds passes.)
DALL-E API returns a result to the oracle backend.
The oracle backend calls the oracle contract in a new transaction, containing the DALL-E response.
The oracle contract calls the callback function in your contract, containing the DALL-E response.

For details on implementation, please see the Solidity reference.

Simple LLM call, with history

Generative image models are stateless: you put a prompt in and get an image out. However, on-chain ChatGPT and many other AI apps require storing state in the form of conversation history. For example, for OpenAI Chat Completions API you need to pass to the API a list of previous messages.

To enable calling LLMs with history, you need to implement two more functions in your contract for fetching history which is stored in your contract, on-chain. Overall, to call an LLM, you need to add the following things to your contract:

Make a call to the LLM. This is done by calling a function on the oracle.
Add a callback function to your contract. This function is called by the oracle when the result is ready.
Add a history getter function to your contract. This function is called by the oracle to get the history. (Detail: this actually needs to be two seperate functions, one for fetching message contents and the other for fetching message roles.)

In this scenario, the execution flow is as follows:

Your contract calls the oracle contract to make a call to the LLM, say OpenAI GPT-4 API.
The oracle backend calls the oracle contract, which in turn calls your contract, to fetch the message history.
The oracle backend makes a call to the LLM API, off-chain.
(About 5-20 seconds passes.)
The LLM API returns a result to the oracle backend.
The oracle backend calls the oracle contract in a new transaction, containing the LLM response.
The oracle contract calls the callback function in your contract, containing the LLM response.
For details on implementation, please see the Solidity reference.

Complex usage: AI agents

AI agents are more complicated. The core of an AI agent is a loop. In each iteration, the output of an LLM call determines what tool to use (e.g. web search), or whether the agent should terminate. After the tool is executed, its output is fed back into the next iteration, where the LLM decides on the next action, etc.

To make an on-chain agent with Galadriel, your callback (that is called when the oracle is ready) needs to contain code for choosing what to do next. For agents, this will typically be one of:

Terminate (if the task is complete, or impossible)
Make a call to an LLM
Make a call to an external tool
This way, the callback function will be repeatedly called by the oracle, until the agent decides to terminate. This creates the loop, even though no explicit loop is present in the contract.

For an example implementation of a simple agent, see this contract.

Security model

A lot of trust is placed in the oracle. This is the first step towards a working system while we are designing and building a minimal-trust approach to this oracle.

In the current architecture of Galadriel, the oracle runs inside a trusted execution environment (TEE) and can be verified by anyone to be executing the correct code; see more in the TEE section.

TEE

The oracle is executed in a trusted execution environment, specifically an AWS Nitro Enclave. Upon startup, the enclave generates a private key which can only be accessed inside the enclave, and is used for signing the transactions made by the oracle. The code running in the enclave posts an attestation onto the chain, proving to the chain that this account (private key) is being used in an enclave with a given image. Since that image, running in the enclave, is public and auditable (see links below), you can be certain that the oracle was correctly executed.

Try out and see teeML in action at teeml.galadriel.com.

Specifically, to trust the oracle and verify everything yourself, you need to:

Trust that the TEE itself is correctly manufactured.
Verify that the enclave image is correct. To do so, you need to
build the oracle docker image,

convert the docker image to a Nitro Enclave image,
verify that the Nitro Enclave image hash is correct (i.e. the hash Galadriel's TEE oracle uses is equal to the hash you received), proving that the enclave image is correctly assembled from the docker image.
Verify the attestation that a TEE-executed oracle is correct, and the enclave image hash included in the attestation matches the hash from step (2.3).
Step (2) is complex and requires nuanced setup in AWS. Step (3) is relatively simple, computationally cheap, and can in principle be done in-browser.

See more details about the TEE setup in the following links.

An interactive explanation of the TEE setup and verification.
Repository containing the enclave setup and instructions for verifying attestations: see galadriel-ai/teeML.
Oracle contract
Oracle back-end code

See also
Implementations of the above on in galadriel-ai/contracts:
Example contracts
Oracle back-end
Oracle contract
Full Solidity reference

TeeML features
Here's a list of main features supported natively by the teeML.

We are constantly adding to this list. If you need a specific model or a tool, please create a feature request in our Github repo and join Discord to discuss!

Try out and see teeML in action at teeml.galadriel.com.

LLMs

Provider	Description
OpenAI	Models supported: GPT-4-turbo, GPT-3.5-turbo
Groq	Models supported: Llama3.1-8B, Llama3.1-80B, Llama3-8B, Llama3-70B, Mixtral-8x7B, Gemma-7B
Anthropic	Models supported: Claude-3.5-Sonnet, Claude-3-Opus, Claude-3-Sonnet, Claude-3-Haiku, Claude-2.1, claude-2.0, claude-instant-1.2

Multimodal

Provider	Description
OpenAI	Models supported: GPT-4o, GPT-4-turbo vision

Text-to-image models

Provider	Description
OpenAI	Models supported: DALL·E 3

Tools

Tool	Description
Google search	Search the web using Google, via Serper API
Code interpreter	Execute Python code using the Bearly code interpreter API
Image generation	Enable LLMs to generate images with DALL·E 3

Tutorials
Calling an LLM: simple
In this tutorial, we'll build a simplified LLM contract using Solidity on Galadriel (Devnet).

We'll use GPT-4-turbo. To use a different LLM like Claude-3.5-Sonnet, Mistral7B, or any other LLM available with teeML follow through with the tutorial and find code references in the end.

To build an LLM with chat history, a.k.a chat bot, visit Calling an LLM: advanced.

Prerequisites

To deploy the contract and interact with it, you will need:

A Galadriel account. For more information on setting up a wallet, visit Setting Up A Wallet.
Some tokens for gas fees. Get Galadriel Devnet tokens from the Faucet.
Go through Quickstart to set up the local dev environment and run the first example.

Contract

Full contract code is below. The same simplified LLM contracts are also available in the contracts folder. We will use GPT-4-turbo example here and go over each variable and function below.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.13;

import "./interfaces/IOracle.sol";

contract OpenAiSimpleLLM {
    address private oracleAddress; // use latest:
https://docs.galadriel.com/oracle-address
    IOracle.Message public message;
    string public response;
    IOracle.OpenAiRequest private config;

    constructor(address initialOracleAddress) {
        oracleAddress = initialOracleAddress;

        config = IOracle.OpenAiRequest({
            model : "gpt-4-turbo", // gpt-4-turbo gpt-4o
            frequencyPenalty : 21, // > 20 for null
            logitBias : "", // empty str for null
            maxTokens : 1000, // 0 for null
            presencePenalty : 21, // > 20 for null
            responseFormat : "{\"type\":\"text\"}",
            seed : 0, // null
            stop : "", // null
            temperature : 10, // Example temperature (scaled up, 10 means 1.0),
> 20 means null
            topP : 101, // Percentage 0-100, > 100 means null
            tools : "",
            toolChoice : "", // "none" or "auto"
            user : "" // null
        });
    }

    function sendMessage(string memory _message) public {
        message = createTextMessage("user", _message);
        IOracle(oracleAddress).createOpenAiLlmCall(0, config);
    }

    // required for Oracle
    function onOracleOpenAiLlmResponse(
        uint /*runId*/,
```

```
        IOracle.OpenAiResponse memory _response,
        string memory _errorMessage
    ) public {
        require(msg.sender == oracleAddress, "Caller is not oracle");
        if (bytes(_errorMessage).length > 0) {
            response = _errorMessage;
        } else {
            response = _response.content;
        }
    }

    // required for Oracle
    function getMessageHistory(
        uint /*_runId*/
    ) public view returns (IOracle.Message[] memory) {
        IOracle.Message[] memory messages = new IOracle.Message[](1);
        messages[0] = message;
        return messages;
    }

    // @notice Creates a text message with the given role and content
    // @param role The role of the message
    // @param content The content of the message
    // @return The created message
    function createTextMessage(string memory role, string memory content)
private pure returns (IOracle.Message memory) {
        IOracle.Message memory newMessage = IOracle.Message({
            role: role,
            content: new IOracle.Content[](1)
        });
        newMessage.content[0].contentType = "text";
        newMessage.content[0].value = content;
        return newMessage;
    }
}
```

Import
To get started, import all of the required interfaces. This gives the oracle the correct data structure.

```
import "./interfaces/IOracle.sol";
```

If you don’t have the IOracle.sollocally, you can also import it via GitHub like this:

```
import
"https://github.com/galadriel-ai/contracts/blob/main/contracts/contracts/
interfaces/IOracle.sol";
```

Variables
The address of the teeML oracle on Galadriel: 0x68EC9556830AD097D661Df2557FBCeC166a0A075.

This needs to be passed in when deploying the contract. For more information on how it works, click here.

```
address private oracleAddress;
```

The message variable will store the user input.

```
IOracle.Message public message; // example: tell me about cats
```

The oracle will store the response in the response variable.

```
string public response; // example: cats are...
```
The config variable will store the LLM configuration. The example uses configuration for OpenAI LLMs.

```
IOracle.OpenAiRequest private config;
```

Constructor
As mentioned, we'll set the oracle address when deploying the contract, so this needs to be an input for the constructor.

We'll also set the LLM config in the constructor. These will be parameters such as model, temperature, seed etc. For more information, visit OpenAIRequest object.

```
constructor(address initialOracleAddress) {
    oracleAddress = initialOracleAddress;

    config = IOracle.OpenAiRequest({
        model : "gpt-4-turbo", // gpt-4-turbo gpt-4o
        frequencyPenalty : 21, // > 20 for null
        logitBias : "", // empty str for null
        maxTokens : 1000, // 0 for null
        presencePenalty : 21, // > 20 for null
        responseFormat : "{\"type\":\"text\"}",
        seed : 0, // null
        stop : "", // null
        temperature : 10, // Example temperature (scaled up, 10 means 1.0), > 20
means null
        topP : 101, // Percentage 0-100, > 100 means null
        tools : "",
        toolChoice : "", // "none" or "auto"
        user : "" // null
    });
}
```

Send Message Function
A public function to initiate the LLM inference. In this example, our function sendMessage accepts a string _message.

Notice the message variable is not sent to the oracle with createOpenAiLlmCall, but rather stored on the same contract. This is done by the helper function createTextMessage.

The oracle will instead read getMessageHistory (next step) from this contract to retrieve the message history. This reading (instead of writing) will save on gas and storage.

createOpenAiLlmCall will call an oracle with the following parameters

0 an id to keep track of the message and it's response (runId in later tutorials)
config set in the constructor

```
function sendMessage(string memory _message) public {
    message = createTextMessage("user", _message);
    IOracle(oracleAddress).createOpenAiLlmCall(0, config);
}

function createTextMessage(string memory role, string memory content) private
pure returns (IOracle.Message memory) {
```

```
    IOracle.Message memory newMessage = IOracle.Message({
        role: role,
        content: new IOracle.Content[](1)
    });
    newMessage.content[0].contentType = "text";
    newMessage.content[0].value = content;
    return newMessage;
}
```

Read Message Function
The getMessageHistory function is used by the orcale to retrieve the message history, or in this case the only user sent message.

In order for the oracle to properly read the message history, the function

Must be called getMessageHistory
Must be a public view function for the oracle to call
Must return type IOracle.Message[]
Include unint parameter to keep track of the user message to respond to (we hardcoded this to 0)

```
function getMessageHistory(
    uint /*_runId*/
) public view returns (IOracle.Message[] memory) {
    IOracle.Message[] memory messages = new IOracle.Message[](1);
    messages[0] = message;
    return messages;
}
```
In this example, the contract is returning a list of messages of length 1. To have a longer message history (for a chat bots), loop through and return a list of messages. For this go to Calling an LLM: advanced.

LLM Response Callback Function
Once the oracle reads the message, it will forward it to teeML to get a response from the LLM.

After LLM returns the response to teeML, it gets pushed on-chain back to oracle, and from there the oracle calls onOracleOpenAiLlmResponse callback function in your contract.

The funciont must be called onOracleOpenAiLlmResponse
Include unint parameter to keep track of the user message to respond to (we hardcoded this to 0)
Includes parameter IOracle.OpenAiResponse with the correct response
Includes parameter errorMessage if an error occurs

```
function onOracleOpenAiLlmResponse(
    uint /*runId*/,
    IOracle.OpenAiResponse memory _response,
    string memory _errorMessage
) public {
    require(msg.sender == oracleAddress, "Caller is not oracle");
    if (bytes(_errorMessage).length > 0) {
        response = _errorMessage;
    } else {
        response = _response.content;
    }
}
```
To ensure that only the oracle contract can send back the response.

```
require(**msg**.sender == oracleAddress, "Caller is not oracle");
```
The contract then sets response to the response from the oracle (or error

message if failed).

Response
After the oracle responds, anyone can access the value by reading the response variable.

```
string response; // tell me about cats
Cats, scientifically known as Felis catus, are…
```

What’s Next?
Congratulations on building your first LLM smart contract.

Explore further:

Instructions on how to deploy the contract on Galadrial Devnet using Hardhat.
Example script to call out deployed contract.
Use other LLMs:
Anthropic Claude-3.5-Sonnet code example
Groq Mistral7B code example
Implement a more advanced LLM chat bot: on-chain ChatGPT..
Dive deeper into the Galadriel documentation, particularly the How It Works section, to understand the underlying technology.
Learn more about other Use Cases to get inspired for your next project.

Tutorials
Calling an LLM: advanced
In this tutorial, we’ll build up a custom ChatGPT-like chatbot step by step using Solidity on the Galadriel Network. If you’re interested in building decentralized AI applications, this guide will help you understand the basics of calling on-chain LLM models and storing conversation history on the blockchain.

Prerequisites
You can read this tutorial as-is to understand the basics of calling an LLM. However, to deploy the contract and interact with it, you will need:

A Galadriel devnet account. For more information on setting up a wallet, visit Setting Up A Wallet.
Some devnet tokens. Get your free devnet tokens from the Faucet.

Create a New Contract
Let’s start by creating a new Solidity file. We’ll incrementally add to that file to build up our chatbot.

```
pragma solidity ^0.8.9;

contract ChatGpt {
    constructor(address initialOracleAddress) {

    }
}
```

Set the Oracle Address
The oracle address is critical for making requests to LLMs — to generate text with an LLM, you need to call the oracle (see How it works for details). Let’s modify the constructor to initialize and store the oracle address in the constructor to ensure your contract can interact with it:

```
address private owner;
address public oracleAddress;

constructor(address initialOracleAddress) {
    owner = msg.sender;
    oracleAddress = initialOracleAddress;
}
```
We also need a function to update the oracle address. As the oracle's address may change, it's necessary to have a mechanism to update it without redeploying the contract. This functionality is encapsulated in the setOracleAddress function, guarded by the onlyOwner modifier to ensure only the contract owner can update it. onlyOracle ensures that only the oracle contract can send back the LLM response to this contract.

```
modifier onlyOwner() {
    require(msg.sender == owner, "Caller is not owner");
    _;
}

modifier onlyOracle() {
    require(msg.sender == oracleAddress, "Caller is not oracle");
    _;
}

event OracleAddressUpdated(address indexed newOracleAddress);

function setOracleAddress(address newOracleAddress) public onlyOwner {
    oracleAddress = newOracleAddress;
    emit OracleAddressUpdated(newOracleAddress);
}
```
Later in this tutorial, when deploying the contract, we will set this address to the concrete address of a Galadriel-provided oracle.

Oracle interface

To interact with the oracle, we need to define an interface. This interface should include the createLlmCall function, which is used to trigger the oracle to make a request to the LLM.

Define this outside the main ChatGpt contract object.

```
interface IOracle {
    function createLlmCall(
        uint promptId
    ) external returns (uint);
}
```

Starting a chat

Now to the core functionality: chatting. We will implement two separate functions: startChat and addMessage. The first is the entry point for a user: it creates a conversation and sends the first message. addMessage is used to add subsequent messages to the conversation.

We need to store the chat history within a conversation somehow — otherwise the chatbot won't be able to remember the context. For this reason we define two structs: a Message struct to store the message content and the role of the sender, and a ChatRun struct to store the conversation history.

```
struct Message {
    string role;
    string content;
```

```
}

struct ChatRun {
    address owner;
    Message[] messages;
    uint messagesCount;
}
```

As you see, the Message struct reflects the message structure used by the OpenAI API and many other compatible APIs.

Given the above definitions, the startChat function initializes a new conversation (ChatRun struct) and adds the first message, from the message argument which will be the end-user's first message. The ChatRun struct is then stored in a mapping, with the chat ID as the key — we need a unique ID for every ChatRun so we can retrieve it again when the oracle makes a callback (which we will implement later).

Finally, we create an LLM call by calling createLlmCall on the oracle, passing the chat ID as an argument. This will trigger the oracle to make a request to the LLM. We also emit an event notifying that a new chat has been created.

```
event ChatCreated(address indexed owner, uint indexed chatId);
mapping(uint => ChatRun) public chatRuns;
uint private chatRunsCount;

function startChat(string memory message) public returns (uint i) {
    ChatRun storage run = chatRuns[chatRunsCount];

    run.owner = msg.sender;
    Message memory newMessage;
    newMessage.content = message;
    newMessage.role = "user";
    run.messages.push(newMessage);
    run.messagesCount = 1;

    uint currentId = chatRunsCount;
    chatRunsCount = chatRunsCount + 1;

    IOracle(oracleAddress).createLlmCall(currentId);
    emit ChatCreated(msg.sender, currentId);

    return currentId;
}
```

Continuing a chat

The addMessage function is used to add subsequent messages to a conversation, once the conversation has been started.

The function first checks if the last message in the conversation was from the assistant (the chatbot). If it wasn't, the function reverts, as the assistant should always respond to the user's messages. The function also checks if the sender is the owner of the chat, as only the chat owner should be able to add messages.

If the checks pass, the function creates a new message and adds it to the conversation. The function then increments the message count and creates the next LLM call by calling createLlmCall on the oracle, passing the chat ID as an argument.

```
function addMessage(string memory message, uint runId) public {
    ChatRun storage run = chatRuns[runId];
    require(
```

```
        keccak256(abi.encodePacked(run.messages[run.messagesCount - 1].role)) ==
keccak256(abi.encodePacked("assistant")),
        "No response to previous message"
    );
    require(
        run.owner == msg.sender, "Only chat owner can add messages"
    );

    Message memory newMessage;
    newMessage.content = message;
    newMessage.role = "user";
    run.messages.push(newMessage);
    run.messagesCount++;
    IOracle(oracleAddress).createLlmCall(runId);
}
```

Message history
Careful readers will have noticed that we never passed in the conversation history (that we stored in the ChatRun object). This is because the message history is not passed in: rather, the oracle fetches the message history from the contract. The oracle does so by calling two functions: getMessageHistoryContents and getMessageHistoryRoles, on your contract, after your contract invokes createLlmCall.

These two methods in combination are used to provide the oracle with the message history necessary for the LLM call. They are called by the oracle after createLlmCall.

The methods should each return a list, one with message contents and the other listing the roles of the message authors. The list lengths should be equal for a given callbackId.

For certain advanced models, such as the gpt-4-turbo, which can handle complex queries including those with image URLs, a more integrated approach is used. This is facilitated through the getMessageHistory method, which provides both the message roles and contents in a structured format suitable for processing by these models.

For example, if the message history is the following:

role content
system You are a helpful assistant
user Hello!
assistant Hi! How can I help?
user How big is the Sun?
Then getMessageHistoryContents should return the following list of 4 items:

```
[
    "You are a helpful assistant",
    "Hello!",
    "Hi! How can I help?",
    "How big is the Sun?"
]
```
…and getMessageHistoryRoles should return the following list of 4 items:

```
["system", "user", "assistant", "user"]
```
Given the above requirements, we can implement the two functions as follows:

```
function getMessageHistoryContents(uint chatId) public view returns (string[]
memory) {
    string[] memory messages = new string[](chatRuns[chatId].messages.length);
```

```
    for (uint i = 0; i < chatRuns[chatId].messages.length; i++) {
        messages[i] = chatRuns[chatId].messages[i].content;
    }
    return messages;
}

function getMessageHistoryRoles(uint chatId) public view returns (string[]
memory) {
    string[] memory roles = new string[](chatRuns[chatId].messages.length);
    for (uint i = 0; i < chatRuns[chatId].messages.length; i++) {
        roles[i] = chatRuns[chatId].messages[i].role;
    }
    return roles;
}
```

Note that the chatId the oracle passes in will be the same ID we passed to createLlmCall when starting the chat.

Oracle callback

At this point, we have implemented everything the oracle needs to process the response on their side. However, we need a way for the oracle to post a response back to our contract. For this, we need to implement a callback function onOracleLlmResponse that the oracle can call once it has processed the response.

The function should take three arguments: the runId (the chat ID we passed in createLlmCall), the response (the response from the LLM), and an errorMessage (non-empty if there was an error). The function should only be callable by the oracle, so we add a onlyOracle modifier to ensure this.

The function first checks if the last message in the conversation was from the user. If it wasn't, the function reverts, as the user should always respond to the assistant's messages. The function then creates a new message with the response from the LLM and adds it to the conversation. The function increments the message count.

```
function onOracleLlmResponse(
    uint runId,
    string memory response,
    string memory /*errorMessage*/
) public onlyOracle {
    ChatRun storage run = chatRuns[runId];
    require(
        keccak256(abi.encodePacked(run.messages[run.messagesCount - 1].role)) ==
keccak256(abi.encodePacked("user")),
        "No message to respond to"
    );

    Message memory newMessage;
    newMessage.content = response;
    newMessage.role = "assistant";
    run.messages.push(newMessage);
    run.messagesCount++;
}
```

Putting it all together

That's it — if you now deploy the contract to the Galadriel Devnet, you can start chatting with your on-chain chatbot. Note that we did not add a system message yet — to customize your chatbot's instructions you can add a system message at the beginning of the conversation history.

You can find the full ChatGPT contract file here. The code in that contract is ordered slightly differently, and contains additions covered in the Retrieval-augmented generation tutorial.

What’s Next?

Congratulations on deploying your on-chain ChatGPT! Explore further:

Implement a more advanced chatbot by adding retrieval-augmented generation to your contract.
Dive deeper into the Galadriel documentation, particularly the How It Works section, to understand the underlying technology.
Experiment with different LLMs, e.g. Groq-hosted open-source LLMS or take control over the nuances of text generation: see Solidity reference and the example contract.
Explore other Use Cases to get inspired for your next project.
Happy building!

Multimodal: vision

In this tutorial we’ll show how to add vision capabilities to the on-chain LLM call (currently available with OpenAI’s gpt-4o and gpt-4-turbo). Simply put this functionality enables the LLM to recognize what is depicted on an image, for example by providing the model with an image and a question “What’s on here?” it is able to provide a descirption about the image.

Prerequisites

This tutorial is written as a continuation to Calling an LLM tutorial. We recommend going through this first as we’re simply changing a few functions here.

Make sure you’re compiler configuration has viaIR set to true. See our config example here.
A Galadriel devnet account. For more information on setting up a wallet, visit Setting Up A Wallet.
Some devnet tokens. Get your free devnet tokens from the Faucet.
Working code from Calling an LLM.

Steps to modify the ChatGPT contract

Change contract object name to OpenAiChatGptVision.

Remove the IOracle interface from code.

```
interface IOracle {
    function createLlmCall(
        uint promptId
    ) external returns (uint);
}
```

Replace it with an import statement.

```
import "./interfaces/IOracle.sol";
```

If this produces an error, it’s because the compiler cannot find IOracle.sol that’s getting imported.

Next to your main contract file create a folder /interfaces and into this add an IOracle.sol file.

Into the newly created IOracle.sol file copy & paste the code from here.

Depending on where you work the filetree should look something like this:

/contracts

```
    /interfaces
        IOracle.sol
    OpenAiChatGptVision.sol
Delete Message struct from the code.


struct Message {
    string role;
    string content;
}
Edit Message parameter in the ChatRun struct.

struct ChatRun {
    address owner;
    IOracle.Message[] messages;
    uint messagesCount;
}
Edit the constructor. Assign the OpenAiRequest configuration in code above
constructor method to config parameter.

IOracle.OpenAiRequest private config;

constructor(address initialOracleAddress) {
    owner = msg.sender;
    oracleAddress = initialOracleAddress;
    chatRunsCount = 0;

    config = IOracle.OpenAiRequest({
        model : "gpt-4-turbo",
        frequencyPenalty : 21, // > 20 for null
        logitBias : "", // empty str for null
        maxTokens : 1000, // 0 for null
        presencePenalty : 21, // > 20 for null
        responseFormat : "{\"type\":\"text\"}",
        seed : 0, // null
        stop : "", // null
        temperature : 10, // Example temperature (scaled up, 10 means 1.0), > 20
means null
        topP : 101, // Percentage 0-100, > 100 means null
        tools : "",
        toolChoice : "", // "none" or "auto"
        user : "" // null
    });
}
Swap out the whole startChat function to the following. This starts the LLM call
with a list of image URLs and text input. In addition to https and base64-
encoded image data URLs, the Oracle supports ipfs:// URLs too.

function startChat(string memory message, string[] memory imageUrls) public
returns (uint i) {
    ChatRun storage run = chatRuns[chatRunsCount];
    run.owner = msg.sender;
    IOracle.Message memory newMessage = IOracle.Message({
        role: "user",
        content: new IOracle.Content[](imageUrls.length + 1)
    });
    newMessage.content[0] = IOracle.Content({
        contentType: "text",
        value: message
    });
    for (uint u = 0; u < imageUrls.length; u++) {
        newMessage.content[u + 1] = IOracle.Content({
            contentType: "image_url",
            value: imageUrls[u]
```

```
        });
    }
    run.messages.push(newMessage);
    run.messagesCount = 1;
    uint currentId = chatRunsCount;
    chatRunsCount = chatRunsCount + 1;
    IOracle(oracleAddress).createOpenAiLlmCall(currentId, config);
    emit ChatCreated(msg.sender, currentId);
    return currentId;
}
```

Swap out the whole onOracleLlmResponse to the onOracleOpenAiLlmResponse function below.

```
function onOracleOpenAiLlmResponse(
    uint runId,
    IOracle.OpenAiResponse memory response,
    string memory errorMessage
) public onlyOracle {
    ChatRun storage run = chatRuns[runId];
    require(
        keccak256(abi.encodePacked(run.messages[run.messagesCount - 1].role)) ==
keccak256(abi.encodePacked("user")),
        "No message to respond to"
    );

    if (!compareStrings(errorMessage, "")) {
        IOracle.Message memory newMessage = IOracle.Message({
            role: "assistant",
            content: new IOracle.Content[](1)
        });
        newMessage.content[0].contentType = "text";
        newMessage.content[0].value = errorMessage;
        run.messages.push(newMessage);
        run.messagesCount++;
    } else {
        IOracle.Message memory newMessage = IOracle.Message({
            role: "assistant",
            content: new IOracle.Content[](1)
        });
        newMessage.content[0].contentType = "text";
        newMessage.content[0].value = response.content;
        run.messages.push(newMessage);
        run.messagesCount++;
    }
}
```

Swap out the addMessage function with the code below.

```
function addMessage(string memory message, uint runId) public {
    ChatRun storage run = chatRuns[runId];
    require(
        keccak256(abi.encodePacked(run.messages[run.messagesCount - 1].role)) ==
keccak256(abi.encodePacked("assistant")),
        "No response to previous message"
    );
    require(
        run.owner == msg.sender, "Only chat owner can add messages"
    );

        IOracle.Message memory newMessage = IOracle.Message({
        role: "user",
        content: new IOracle.Content[](1)
    });
    newMessage.content[0].contentType = "text";
    newMessage.content[0].value = message;
```

```
    run.messages.push(newMessage);
    run.messagesCount++;

    IOracle(oracleAddress).createOpenAiLlmCall(runId, config);
}
```

Let’s also refactor the getMessageHistory function to the following.

```
function getMessageHistory(uint chatId) public view returns (IOracle.Message[]
memory) {
    return chatRuns[chatId].messages;
}
```

Finally add one helper function to the end.

```
function compareStrings(string memory a, string memory b) private pure returns
(bool) {
    return (keccak256(abi.encodePacked((a))) ==
keccak256(abi.encodePacked((b))));
}
```

Putting it all together

That’s it — if you now deploy the contract to the Galadriel Devnet, you can start sending image url’s with text as input to the on-chain LLM.

You can find the full OpenAiChatGptVision contract file here. The code in that contract is ordered slightly differently.

What’s Next?

Congratulations on deploying your on-chain ChatGPT! Explore further:

Implement a more advanced chatbot by adding retrieval-augmented generation to your contract.
Dive deeper into the Galadriel documentation, particularly the How It Works section, to understand the underlying technology.
Experiment with different LLMs, e.g. Groq-hosted open-source LLMS or take control over the nuances of text generation: see Solidity reference and the example contract.
Explore other Use Cases to get inspired for your next project.
Happy building!

Tutorials

Retrieval-augmented generation

In this tutorial, we’ll enhance a ChatGPT-like chatbot with a knowledge base. The knowledge base will be used to retrieve relevant information that can be used to generate more informative and contextually relevant responses. This is also known as retrieval-augmented generation (RAG).

Prerequisites

You can read this tutorial as-is to understand the basics of RAG with Galadriel. However, to create a knowledge base you can use in your contract, you will need:

Python 3.11 or later installed on your system.
A Galadriel devnet account. For more information on setting up a wallet, visit Setting Up A Wallet.
Some devnet tokens. Get your free devnet tokens from the Faucet.
An API key (JWT) for pinata.cloud to facilitate document uploading to IPFS. You can obtain this key by registering at pinata.cloud.
The files you want to add to the knowledge base (see below for supported formats).

What is RAG?

RAG is a solution to a simple problem. Assume you want to make an app that can answer questions based on a particular book. To make an LLM answer questions based on a book, you need the relevant parts of the book to be available to the model, so you need to put these parts into the context window (input prompt to the LLM). However, since input tokens are expensive, you don't want to put in the whole book on every query.

RAG solves this problem by storing the book in a knowledge base and only retrieving the relevant parts of the book when needed. The straightforward implementation of this is to:

Divide the book into smaller parts (e.g., paragraphs).
Put these paragraphs into an index structure for fast retrieval.
When a query comes in, retrieve the relevant paragraphs from the index and put them into the context window.
Galadriel's RAG implementation does this and uses the industry-standard approach of creating document embeddings, specifically using OpenAI's text-embedding-3-small model, to support semantic search (as opposed to simple keyword-based search).

The key parts of the RAG implementation are:

Collecting the text: you upload to IPFS a list of text documents that will comprise the knowledge base.
Building the index: once you give the IPFS link (pointing to a list of documents), our oracle running in a TEE will embed these documents and build an index.
Querying the index: you can query the index by calling a function in your contract, including a reference to the index in IPFS. The oracle will retrieve the relevant documents and return them to you.
This way, the knowledge base is stored off-chain in IPFS. However, a reference to the knowledge base is stored on-chain. This way it is guaranteed to be used correctly because the oracle runs in a TEE, and the IPFS file cannot be modified (otherwise the CID will change)

Repository and environment setup

Here's how you can setup your environment to create a knowledge base.

1
If you haven't already, clone the Galadriel repository:

```
git clone https://github.com/galadriel-ai/contracts.git
cd contracts/rag_tools
```

2
Create a virtual environment for Python dependencies:

```
python -m venv venv
source venv/bin/activate
```

3
Install the required Python packages:

```
pip install -r requirements.txt
```

4
Create a .env file and add your pinata.cloud JWT and wallet private key as follows:

```
ORACLE_ADDRESS=galadriel_oracle_address
PRIVATE_KEY=your_wallet_private key
```

PINATA_API_KEY=your_api_key_here
ORACLE_ADDRESS should be set to the current Galadriel oracle address: 0x68EC9556830AD097D661Df2557FBCeC166a0A075.

PRIVATE_KEY will be used to make an indexing request to the oracle. You can use any account as long as it has enough devnet tokens to pay for the transaction.

PINATA_API_KEY will be used to upload the documents to IPFS.

Collect your knowledge base files
Collect all files you want to end up in your knowledge base in a single directory. For example, into a kb folder:

```
$ ls kb/
bitcoin.pdf
ethereum.pdf
```

You can put anything you want into the knowledge base, as long as it can be converted into text format. The scripts we provide uses the Unstructured file loader in langchain to do a default conversion for you for a list of common formats: text files, powerpoints, html, pdfs, images, and more. However, you can customize the extraction process by customizing the loader in add_knowledge_base.py script.

Index the knowledge base
Now we can execute the script that will extract the text from the documents in the kb/ directory and upload them to IPFS. After uploading the documents, the script will request the oracle to index the documents. Once the index is complete, the script will output the IPFS hash of the index.

The execution of the script should look like this:

```
$ python add_knowledge_base.py -d kb -s 1500
[Loading 2 files from kb.]
Processing Files: 100%|████████████████████| 2/2 [00:08<00:00,  4.19s/file]
Generated 78 documents from 2 files.
Uploading documents to IPFS, please wait...done.
Requesting indexing, please wait...done.
Waiting for indexing to complete...done.
Knowledge base indexed, index CID
`QmdVEfixTc7qf7HQLN6UqxPsiMKSDcdKx3fSjiri2MRAfR`.
Use CID `bafkreiduon46ccwwteicuqwjwikd43f7mt7xzpp6kjdheducgzoex555xa` in your
contract to query the indexed knowledge base.
Make sure you store the index IPFS CID from this output as we will need it later
— in this example it is
bafkreiduon46ccwwteicuqwjwikd43f7mt7xzpp6kjdheducgzoex555xa.
```

Integrating in your contract
Now that your knowledge base is indexed, we can actually query it in a contract. To do this, we need to do two things:

Store the index IPFS hash in the contract.
Make a call to the oracle to start the knowledge base query: calling createKnowledgeBaseQuery.
Add a listener to our contract which the oracle will call once the results of the query are ready: implementing onOracleKnowledgeBaseQueryResponse.
We will go through these steps in the next sections. See the oracle reference for a full description of the RAG-related interface.

Store the index IPFS hash in the contract
Since on every query we need to reference the index, we need to store the index IPFS hash in the contract. This can be done by adding a new state variable to the contract:

```
string public knowledgeBase;
```

You might also want to add this into the constructor of your contract:

```
constructor(address initialOracleAddress, string memory knowledgeBaseCID) {
    owner = msg.sender;
    oracleAddress = initialOracleAddress;
    knowledgeBase = knowledgeBaseCID;
}
```

At contract deployment time, you will pass in the IPFS CID of the specific knowledge base you created above.

Create a knowledge base query
In your contract, at the point where you want to query the knowledge base, you need to call the createKnowledgeBaseQuery function in the oracle. This function takes the IPFS link to the index as an argument. The oracle will then retrieve the relevant documents and return them to you.

Make sure you have the oracle interface defined in your contract file:

```
interface IOracle {
    function createKnowledgeBaseQuery(
        uint kbQueryCallbackId,
        string memory cid,
        string memory query,
        uint32 num_documents
    ) external returns (uint i);
}
```

You need to call the createKnowledgeBaseQuery function immediately prior to every call to the LLM (because on every LLM call we want the knowledge base contents to be available). If you've built your ChatGPT-like chatbot similarly to our example, that means adding a call to createKnowledgeBaseQuery in two places: startChat and addMessage. Refer to the chatbot tutorial for a full explanation of these methods.

In both cases, you should make sure the knowledge base IPFS hash is non-empty. There's another tricky thing: if there is no knowledge base, you want to call the LLM immediately. If there is one, you need to call the LLM in the onOracleKnowledgeBaseQueryResponse callback. (You can simplify this if you always deploy your contract with a knowledge base and it is never empty.)

```
if (bytes(knowledgeBase).length > 0) {
    // If there is a knowledge base, create a knowledge base query
    IOracle(oracleAddress).createKnowledgeBaseQuery(
        runId,
        knowledgeBase,
        message,
        3
    );
} else {
    // Otherwise, create an LLM call
    IOracle(oracleAddress).createLlmCall(runId);
}
```

In the snippet above, message is the text used to query documents in the knowledge base.

Implement the callback
Once the knowledge base query is complete, the oracle will call the onOracleKnowledgeBaseQueryResponse function in your contract. This function should be implemented to handle the results of the query. The results will be a list of plaintext documents.

See the inline comments below for an explanation of this function — which again assumes you are using the ChatGPT-like chatbot from our example.

```
function onOracleKnowledgeBaseQueryResponse(
    uint runId,
    string [] memory documents,
    string memory errorMessage
) public onlyOracle {
    ChatRun storage run = chatRuns[runId];
    require(
        keccak256(abi.encodePacked(run.messages[run.messagesCount - 1].role)) ==
keccak256(abi.encodePacked("user")),
        "No message to add context to"
    );
    // Retrieve the last user message
    Message storage lastMessage = run.messages[run.messagesCount - 1];

    // Start with the original message content
    string memory newContent = lastMessage.content;

    // Append "Relevant context:\n" only if there are documents
    if (documents.length > 0) {
        newContent = string(abi.encodePacked(newContent, "\n\nRelevant context:\
n"));
    }

    // Iterate through the documents and append each to the newContent
    for (uint i = 0; i < documents.length; i++) {
        newContent = string(abi.encodePacked(newContent, documents[i], "\n"));
    }

    // Finally, set the lastMessage content to the newly constructed string
    lastMessage.content = newContent;

    // Call LLM
    IOracle(oracleAddress).createLlmCall(runId);
}
```

Putting it all together
That’s it — if you now deploy the contract to the Galadriel Devnet, you can start chatting with your on-chain chatbot.

You’ll find a fully functional KB-enabled chatbot in our example contracts directory: ChatGpt.sol.

What’s Next?
Congratulations on deploying your RAG-enabled chatbot! Explore further:

Dive deeper into the Galadriel documentation, particularly the How It Works section, to understand the underlying technology.
Review the full RAG reference to understand the full capabilities of the RAG oracle.
Experiment with different LLMs, e.g. Groq-hosted open-source LLMS or take control over the nuances of text generation: see Solidity reference.

Explore other Use Cases to get inspired for your next project.
Happy building!

Agents

Introduction

AI agents are a specific kind of AI application. They are mainly defined by two things: an LLM running in a loop and access to the external world via tools. An AI agent is in effect an LLM that is repeatedly asked to make a decision: given a user input and progress so far, what should be done next? Giving LLMs reasoning capabilities, the architecture has proven powerful and flexible, and is used for a variety of applications.

A common example is a researcher AI agent. Given tools for searching the web, browsing websites, and perhaps storing outputs in a scratch file, the agent assembles a report on a particular topic. After every loop the LLM decides the next action (use a tool or any other custom action defined by the dev), given the progress it’s had so far.

Image by Alex Honchar

In this tutorial, our goal will be to create a simple agent that can create images based on live information, e.g. the current weather in the user’s chosen location.

Try out and see the agent in action by running the agent.ts script that calls already deployed contract.

Deployed contract address: 0x9496b155e6e2BCdf5aC0d64F36d344E5B8e49d60

Prerequisites

You can read this tutorial as-is to understand the basics of building an agent. However, to deploy the contract and interact with it, you will need:

A Galadriel devnet account. For more information on setting up a wallet, visit Setting Up A Wallet.
Some devnet tokens. Get your free devnet tokens from the Faucet.

Before agents: single LLM call

As the first step, let’s create a contract that makes a single LLM call without any loop or external tools. Once we have that contract in place, we’ll build on it until we have a complete agent with loops.

First, set up the basic smart contract with an oracle integration point.

```
pragma solidity ^0.8.9;

import "./interfaces/IOracle.sol";

contract Agent {
    address private owner;
    address public oracleAddress;
    string public prompt;

    event OracleAddressUpdated(address indexed newOracleAddress);

    constructor(address initialOracleAddress, string memory systemPrompt) {
        owner = msg.sender;
        oracleAddress = initialOracleAddress;
```

```
        prompt = systemPrompt;
    }

    modifier onlyOwner() {
        require(msg.sender == owner, "Caller is not owner");
        _;
    }

    function setOracleAddress(address newOracleAddress) public onlyOwner {
        oracleAddress = newOracleAddress;
        emit OracleAddressUpdated(newOracleAddress);
    }
}
```

The constructor includes a systemPrompt parameter, which sets the foundational behavior of your agent. Specifying this prompt at the time of deploying the smart contract enhances flexibility, allowing the agent's behavior to be tailored from the outset. A typical example of a system prompt might be You are a helpful assistant. This prompt guides the agent's interactions, ensuring it consistently behaves in a helpful and assistant-like manner.

A simple LLM call

Our first task is to establish a foundation by creating a smart contract that can make a single LLM call. This initial step helps us understand the basic interaction with an LLM without looping or using external tools.

Configuration

Since we are going to use OpenAI LLM, first step is to define configuration parameters using OpenAiRequest object. We will do that in smart contract constructor.

```
IOracle.OpenAiRequest private config;

constructor(address initialOracleAddress, string memory systemPrompt) {
    owner = msg.sender;
    oracleAddress = initialOracleAddress;
    prompt = systemPrompt;

    config = IOracle.OpenAiRequest({
        model : "gpt-4-turbo-preview",
        frequencyPenalty : 21, // > 20 for null
        logitBias : "", // empty str for null
        maxTokens : 1000, // 0 for null
        presencePenalty : 21, // > 20 for null
        responseFormat : "{\"type\":\"text\"}",
        seed : 0, // null
        stop : "", // null
        temperature : 10, // Example temperature (scaled up, 10 means 1.0), > 20
means null
        topP : 101, // Percentage 0-100, > 100 means null
        tools : "",
        toolChoice : "",
        user : "" // null
    });
}
```

At the moment we don't need any tools defined.

Making an LLM call

We need to store the agent steps within a run. For this reason we define two structs: a Message struct to store the message content and the role of the

sender, and an AgentRun struct to store the agent history.

```
struct Message {
    string role;
    string content;
}
struct AgentRun {
    address owner;
    Message[] messages;
    uint responsesCount;
    uint8 max_iterations;
    bool is_finished;
}
```

Given the above definitions, the runAgent function initializes a new agent run (AgentRun struct) and adds the system prompt and first message, from the message argument which will be the end-user's first message. The AgentRun struct is then stored in a mapping, with the run ID as the key — we need a unique ID for every Agent so we can retrieve it again when the oracle makes a callback (which we will implement later). max_iterations parameter is used to define how many loops the agent can do before it stops.

Finally, we create an LLM call by calling createOpenAiLlmCall on the oracle, passing the OpenAiRequest configuration and run ID as an argument. This will trigger the oracle to make a request to the LLM. We also emit an event notifying that a new agent run has been created.

```
event AgentRunCreated(address indexed owner, uint indexed runId);
mapping(uint => AgentRun) public agentRuns;
uint private agentRunCount;

function runAgent(string memory query, uint8 max_iterations) public returns
(uint i) {
    AgentRun storage run = agentRuns[agentRunCount];

    run.owner = msg.sender;
    run.is_finished = false;
    run.responsesCount = 0;
    run.max_iterations = max_iterations;

    Message memory systemMessage;
    systemMessage.content = prompt;
    systemMessage.role = "system";
    run.messages.push(systemMessage);

    Message memory newMessage;
    newMessage.content = query;
    newMessage.role = "user";
    run.messages.push(newMessage);

    uint currentId = agentRunCount;
    agentRunCount = agentRunCount + 1;
    IOracle(oracleAddress).createOpenAiLlmCall(currentId, config);

    emit AgentRunCreated(run.owner, currentId);

    return currentId;
}
```

The oracle needs to fetch the agent history from the contract. It does so by calling two functions: getMessageHistoryContents and getMessageHistoryRoles, on your contract, after your contract invokes createOpenAiLlmCall.

The methods should each return a list, one with message contents and the other

listing the roles of the message authors. The list lengths should be equal for a given callbackId.

```
function getMessageHistoryContents(uint agentId) public view returns (string[]
memory) {
    string[] memory messages = new string[](agentRuns[agentId].messages.length);
    for (uint i = 0; i < agentRuns[agentId].messages.length; i++) {
        messages[i] = agentRuns[agentId].messages[i].content;
    }
    return messages;
}
function getMessageHistoryRoles(uint agentId) public view returns (string[]
memory) {
    string[] memory roles = new string[](agentRuns[agentId].messages.length);
    for (uint i = 0; i < agentRuns[agentId].messages.length; i++) {
        roles[i] = agentRuns[agentId].messages[i].role;
    }
    return roles;
}
```

We need a way for the oracle to post a response back to our contract. For this, we implement a callback function onOracleLlmResponse that the oracle can call once it has processed the response. The function should take three arguments: the runId (the run ID we passed in createLlmCall), the response (the response from the LLM), and an errorMessage (non-empty if there was an error). The function should only be callable by the oracle, so we add a onlyOracle modifier to ensure this.

The function first checks if the errorMessage is non-empty. If it wasn’t, the function adds the error to the agent history and marks the run as finished. Otherwise it creates a new message with the response from the LLM and adds it to the agent history. The function increments the message count. Since our goal is to make just one LLM call, the function is marking the run as finished.

```
function onOracleOpenAiLlmResponse(
    uint runId,
    IOracle.OpenAiResponse memory response,
    string memory errorMessage
) public onlyOracle {
    AgentRun storage run = agentRuns[runId];
    if (!compareStrings(errorMessage, "")) {
        Message memory newMessage;
        newMessage.role = "assistant";
        newMessage.content = errorMessage;
        run.messages.push(newMessage);
        run.responsesCount++;
        run.is_finished = true;
        return;
    }
    if (!compareStrings(response.content, "")) {
        Message memory assistantMessage;
        assistantMessage.content = response.content;
        assistantMessage.role = "assistant";
        run.messages.push(assistantMessage);
        run.responsesCount++;
    }
    run.is_finished = true;
}
```

Making a loop

So far we’ve made an agent that makes one LLM query and finishes the run. We will now try to modify our agent to make one extra LLM call after getting results from a previous one, thus creating an agent loop.

Let’s modify our onOracleOpenAiLlmResponse to do so. Upon successful response from the Oracle, the agent will ask for more details. We can track agent steps by checking the responsesCount variable and make a decision about whether the agent should make another loop (as the result is incomplete) or end the run.

```
function onOracleOpenAiLlmResponse(
    uint runId,
    IOracle.OpenAiResponse memory response,
    string memory errorMessage
) public onlyOracle {
    AgentRun storage run = agentRuns[runId];
    if (!compareStrings(errorMessage, "")) {
        Message memory assistantMessage;
        assistantMessage.role = "assistant";
        assistantMessage.content = errorMessage;
        run.messages.push(assistantMessage);
        run.responsesCount++;
        run.is_finished = true;
        return;
    }
    if (!compareStrings(response.content, "")) {
        Message memory assistantMessage;
        assistantMessage.content = response.content;
        assistantMessage.role = "assistant";
        run.messages.push(assistantMessage);
        run.responsesCount++;

        // if this is the first response, ask for more details
        if (run.responsesCount == 1) {
            Message memory newMessage;
            newMessage.content = "Please elaborate!";
            newMessage.role = "user";
            run.messages.push(newMessage);
            IOracle(oracleAddress).createOpenAiLlmCall(runId, config);
            return;
        }
        // we already asked for clarification, mark the agent run as finished
        run.is_finished = true;
    }
}
```

Example run with an LLM loop
1
Execute the following command to run a script that calls the deployed contract:

```
npm run agent
```

The script will interactively ask for the input and then call the contract with the input. The output, when ready, will be printed to the console.

If this step fails, make sure you have set the AGENT_ADDRESS environment variable to the address of your deployed contract.

```
Agent's task: Who is the president of United States?
Max iterations: 5
Task sent, tx hash:
0x20418bcd17d5c8714c59f4ccab02c8d45eb9c368523f53f45e1eb58af3296c03
Agent started with task: "Who is the president of United States?"
Created agent run ID: 0
STEP: You are a helpful assistant
STEP: Who is the president of United States?
```

THOUGHT: As of my last update in 2023, Joe Biden is the President of the United States. He was inaugurated as the 46th president on January 20, 2021.
STEP: Please elaborate!
THOUGHT: Joe Biden, whose full name is Joseph Robinette Biden Jr., is an American politician from the Democratic Party who assumed office as the 46th President of the United States on January 20, 2021. Before becoming president, Joe Biden had a long and distinguished career in public service. He was born on November 20, 1942, in Scranton, Pennsylvania, and later moved to Delaware.

Biden's early entry into politics saw him elected to the New Castle County Council in 1970. Just two years later, at the age of 29, he made a significant leap into national politics by winning a seat in the U.S. Senate, representing Delaware. He would go on to serve in the Senate for 36 years, making him one of the longest-serving senators in American history. During his tenure in the Senate, Biden was known for his work on foreign relations, criminal justice, and drug policy. He served as the Chairman of the Senate Judiciary Committee and was also the Chairman of the Senate Foreign Relations Committee for several years.

In 2008, Joe Biden was elected Vice President of the United States as the running mate of Barack Obama. He served two terms as Vice President from 2009 to 2017. Their administration was notable for its efforts in healthcare reform, economic recovery following the Great Recession, and foreign policy achievements, including the operation that led to the death of Osama bin Laden.

After briefly retiring from public office following his vice presidency, Biden announced his candidacy for the 2020 presidential election. He won the Democratic nomination and went on to defeat the incumbent, Donald Trump, in the November 2020 election. The Biden-Harris ticket’s victory was historic for several reasons, including Kamala Harris becoming the first female Vice President, as well as the highest-ranking female official in U.S. history, and the first African American and Asian American Vice President.

President Biden's administration has focused on addressing the COVID-19 pandemic, economic recovery, climate change, and civil rights. Biden has emphasized unity and healing for the nation, aiming to bridge the political divide and address the challenges facing America in the 21st century.
agent run ID 0 finished!
Done

Adding a tool: web search
Let’s add a web search tool to the agent’s capabilities, allowing it to retrieve real-time data such as the current weather.

We first modify our OpenAiRequest configuration by defining the web_search tool and setting toolChoice to auto.

```
    config = IOracle.OpenAiRequest({
        //...
        tools : "[{\"type\":\"function\",\"function\":
{\"name\":\"web_search\",\"description\":\"Search the internet\",\"parameters\":
{\"type\":\"object\",\"properties\":{\"query\":
{\"type\":\"string\",\"description\":\"Search query\"}},\"required\":
[\"query\"]}}}]",
        toolChoice : "auto", // "none" or "auto"
        //...
    });
```

This setup configures the OpenAI LLM to utilize the web_search tool as needed. However, the responsibility to initiate the tool usage lies with us. To facilitate this, we will adjust the onOracleOpenAiLlmResponse function. This function will not only check when a tool is required and activate it accordingly, but it will also monitor whether the agent run has reached its predefined limit for loops or has concluded with a complete satisfying result.

```
    function onOracleOpenAiLlmResponse(
        uint runId,
        IOracle.OpenAiResponse memory response,
        string memory errorMessage
    ) public onlyOracle {
        AgentRun storage run = agentRuns[runId];
        require(
            !run.is_finished, "Run is finished"
        );

        if (!compareStrings(errorMessage, "")) {
            Message memory newMessage;
            newMessage.role = "assistant";
            newMessage.content = errorMessage;
            run.messages.push(newMessage);
            run.responsesCount++;
            run.is_finished = true;
            return;
        }
        if (run.responsesCount >= run.max_iterations) {
            run.is_finished = true;
            return;
        }
        if (!compareStrings(response.content, "")) {
            Message memory assistantMessage;
            assistantMessage.content = response.content;
            assistantMessage.role = "assistant";
            run.messages.push(assistantMessage);
            run.responsesCount++;
        }
        if (!compareStrings(response.functionName, "")) {
            IOracle(oracleAddress).createFunctionCall(runId,
response.functionName, response.functionArguments);
            return;
        }
        // the LLM has given the answer, nothing else to do
        run.is_finished = true;
    }
```

To manage the output from a tool call effectively, we need to ensure its reception from the oracle is handled appropriately. We will implement the onOracleFunctionResponse function to receive either the output from the tool or an error message. After capturing and storing this information in the agent’s history, we will initiate another LLM call to process the results, continuing the agent’s operations seamlessly.

```
function onOracleFunctionResponse(
        uint runId,
        string memory response,
        string memory errorMessage
    ) public onlyOracle {
        AgentRun storage run = agentRuns[runId];
        require(
            !run.is_finished, "Run is finished"
        );
        string memory result = response;
        if (!compareStrings(errorMessage, "")) {
            result = errorMessage;
        }
        Message memory newMessage;
        newMessage.role = "user";
        newMessage.content = result;
        run.messages.push(newMessage);
```

```
        run.responsesCount++;
        IOracle(oracleAddress).createOpenAiLlmCall(runId, config);
    }
```

Example run with web_search tool

```
Agent started with task: "What is the weather like in New York?"
Created agent run ID: 0
STEP: You are a helpful assistant
STEP: What is the weather like in New York?
STEP: [{"title": "Weather Forecast and Conditions for New York City, NY",
"link":
"https://weather.com/weather/today/l/96f2f84af9a5f5d452eb0574d4e4d8a840c71b05e22
264ebdc0056433a642c84", "snippet": "New York City, NY. As of 7:54 am EDT. 46\
u00b0. Drizzle. Day 49\u00b0 \u2022 Night 45\u00b0. Small Craft Advisory. Latest
News. Why 2024 Will Be One Of The Most Active Seasons ...", "position": 1},
{"title": "Hourly Weather Forecast for New York City, NY", "link":
"https://weather.com/weather/hourbyhour/l/f892433d7660da170347398eb8e3d722d8d362
fe7dd15af16ce88324e1b96e70", "snippet": "Wednesday, April 17 \u00b7 1 am. 57\
u00b0. 1%. Partly Cloudy. Feels Like57\u00b0. WindNNE 3 mph. Humidity46%. UV
Index0 of 11 \u00b7 2 am. 56\u00b0. 2%. Mostly Clear. Feels Like56\u00b0.",
"position": 2}, {"title": "Current Weather - New York, NY - AccuWeather",
"link": "https://www.accuweather.com/en/us/new-york/10021/current-weather/
349727", "snippet": "New York, NY \u00b7 Current Weather. 7:15 AM. 47\u00b0F.",
"position": 3}, {"title": "Hourly Weather Forecast for Manhattan, NY", "link":
"https://weather.com/weather/hourbyhour/l/Manhattan+NY?
canonicalCityId=fc47c333c5d13e34e34c9fdb6e047ceb70f7891e01bc9e1d574b5f93f58aa76d
", "snippet": "Friday, April 19 \u00b7 1 am. 47\u00b0. 4%. Mostly Cloudy. Feels
Like45\u00b0. WindNE 5 mph. Humidity65% \u00b7 2 am. 47\u00b0. 4%. Mostly
Cloudy. Feels Like45\u00b0. WindENE 5 mph. Humidity ...", "position": 4},
{"title": "New York, NY Weather Forecast - AccuWeather", "link":
"https://www.accuweather.com/en/us/new-york/10021/weather-forecast/349727",
"snippet": "10-Day Weather Forecast ; Today. 4/18. 50\u00b0 44\u00b0 ; Fri.
4/19. 56\u00b0 48\u00b0 ; Sat. 4/20. 67\u00b0 45\u00b0 ; Sun. 4/21. 60\u00b0 46\
u00b0 ; Mon. 4/22. 65\u00b0 49\u00b0.", "sitelinks": [{"title": "Current
Weather", "link": "https://www.accuweather.com/en/us/new-york/10021/current-
weather/349727"}, {"title": "Daily", "link":
"https://www.accuweather.com/en/us/new-york/10021/daily-weather-forecast/
349727"}, {"title": "Hourly", "link": "https://www.accuweather.com/en/us/new-
york/10021/hourly-weather-forecast/349727"}, {"title": "MinuteCast", "link":
"https://www.accuweather.com/en/us/new-york/10021/minute-weather-forecast/
349727"}], "position": 5}, {"title": "Current WeatherNew York, NY - The Weather
Network", "link": "https://www.theweathernetwork.com/en/city/us/new-york/new-
york/current", "snippet": "Current WeatherNew York, NY. New York, NY. Updated 7
minutes ago. 8. \u00b0C. Light rain. Feels 4. H: 10\u00b0 L: 7\u00b0. Hourly.
Full 72 hours \u00b7 12pm. Cloudy. 8\u00b0.", "sitelinks": [{"title": "Hourly",
"link": "https://www.theweathernetwork.com/us/hourly-weather-forecast/new-york/
new-york"}, {"title": "14 Day", "link":
"https://www.theweathernetwork.com/us/14-day-weather-trend/new-york/new-york"},
{"title": "7 Day", "link": "https://www.theweathernetwork.com/us/36-hour-
weather-forecast/new-york/new-york"}], "position": 6}, {"title": "Weather
Forecast and Conditions for Manhattan, NY", "link":
"https://weather.com/weather/today/l/New+York+NY+USNY0996", "snippet":
"Manhattan, NY. As of 11:01 am EDT. 61\u00b0. Partly Cloudy. Day 61\u00b0 \u2022
Night 48\u00b0. Small Craft Advisory. Latest News. 'Everything's Gone'- Tornado
Survivor ...", "position": 7}, {"title": "Accutrack Radar | New York Weather
News - abc7NY", "link": "https://abc7ny.com/weather/", "snippet": "Today's
Weather. New York City, NY. Current; Tonight; Tomorrow. CLOUDY. 50\u00b0. Feels
Like. 45\u00b0. Sunrise. 6:13 AM. Humidity. 81%. Sunset. 7:38 PM. Windspeed. ENE
...", "sitelinks": [{"title": "AccuWeather: Raw and damp", "link":
"https://abc7ny.com/nyc-weather-accuweather-forecast-new-york-city-tri-state/
27410/"}, {"title": "AccuTrack Radar", "link":
"https://abc7ny.com/weather/doppler/accutrack-radar/"}, {"title": "Weather
Alerts", "link": "https://abc7ny.com/weather/alerts/"}, {"title": "New Jersey",
```

"link": "https://abc7ny.com/weather/doppler/new-jersey/"}], "position": 8},
{"title": "Weather - NBC New York", "link":
"https://www.nbcnewyork.com/weather/", "snippet": "Click to search by city or
zip... New York City, NY. 47\u00b0. Light Drizzle. Feels like40\u00b0.
Humidity90%. Precip0.09%. Wind16MPH. Temperature Precipitation. 47\u00ba.",
"sitelinks": [{"title": "Maps and Radar", "link":
"https://www.nbcnewyork.com/weather/maps/"}, {"title": "Weather", "link":
"https://www.nbcnewyork.com/tag/weather/"}, {"title": "Weather Alerts", "link":
"https://www.nbcnewyork.com/weather/severe-weather-alerts/"}, {"title": "School
Closings", "link": "https://www.nbcnewyork.com/weather/school-closings/"}],
"position": 9}, {"title": "New York City, NY Hourly Weather Forecast", "link":
"https://www.wunderground.com/hourly/us/ny/new-york-city", "snippet": "New York
City, NY Hourly Weather Forecaststar_ratehome ; 1:00 pm, Cloudy, 46 \u00b0F ;
2:00 pm, Cloudy, 47 \u00b0F ; 3:00 pm, Cloudy, 48 \u00b0F ; 4:00 pm, Cloudy,
48 \u00b0F ...", "position": 10}]
THOUGHT: The current weather in New York City, NY is as follows:

- **Temperature**: 46°F (7.8°C) with drizzle.
- **Day Temperature**: Up to 49°F.
- **Night Temperature**: Around 45°F.
- **Wind**: Northeast with speeds around 3 to 5 mph.
- **Humidity**: Ranges from 46% to around 65%.
- **Conditions**: Mainly drizzle and cloudy conditions across different parts of the city.

For more detailed conditions, including forecasts and advisories, you may visit the provided links.
agent run ID 0 finished!
Done
It’s evident that the agent utilized the web_search tool, incorporated the results into its history, and then proceeded to invoke the LLM, which produced the final response.

Adding a tool: image generation
To integrate an additional tool, simply include it in the OpenAiRequest configuration. No further modifications to the agent’s logic are required, as the process is designed to be agnostic of the specific tool being used.

```
config = IOracle.OpenAiRequest({
    //...
    tools : "[{\"type\":\"function\",\"function\":
{\"name\":\"web_search\",\"description\":\"Search the internet\",\"parameters\":
{\"type\":\"object\",\"properties\":{\"query\":
{\"type\":\"string\",\"description\":\"Search query\"}},\"required\":
[\"query\"]}}},{\"type\":\"function\",\"function\":
{\"name\":\"image_generation\",\"description\":\"Generates an image using Dalle-
2\",\"parameters\":{\"type\":\"object\",\"properties\":{\"prompt\":
{\"type\":\"string\",\"description\":\"Dalle-2 prompt to generate an
image\"}},\"required\":[\"prompt\"]}}}]",
    //...
});
```

Example run with image_generation tool

```
Agent's task: Generate an image of a beaver.
Max iterations: 5
Task sent, tx hash:
0xd3e27b24cac0417243642599a917f075d89ceaf6e6aea81ccaab1ba9dcd1347d
Agent started with task: "Generate an image of a beaver."
Created agent run ID: 0
STEP: You are a helpful assistant
STEP: Generate an image of a beaver.
```

STEP: https://oaidalleapiprodscus.blob.core.windows.net/private/org-OIsLdcbHIc1F6swhIoWbkFsS/user-dvcNG9YUCv8srEVF1LN250bv/img-Nrhz2ojGrZXBkWoOMnvZrYXx.png?st=2024-04-18T18%3A24%3A06Z&se=2024-04-18T20%3A24%3A06Z&sp=r&sv=2021-08-06&sr=b&rscd=inline&rsct=image/png&skoid=6aaadede-4fb3-4698-a8f6-684d7786b067&sktid=a48cca56-e6da-484e-a814-9c849652bcb3&skt=2024-04-17T22%3A32%3A23Z&ske=2024-04-18T22%3A32%3A23Z&sks=b&skv=2021-08-06&sig=AH5PvmftgbhhZHm1ZyCUtcUw%2BdboorxzV%2BDRUruIHVM%3D
THOUGHT: I've generated an image of a beaver for you. You can view the image by clicking on the link below:

[View Beaver Image](https://oaidalleapiprodscus.blob.core.windows.net/private/org-OIsLdcbHIc1F6swhIoWbkFsS/user-dvcNG9YUCv8srEVF1LN250bv/img-Nrhz2ojGrZXBkWoOMnvZrYXx.png?st=2024-04-18T18%3A24%3A06Z&se=2024-04-18T20%3A24%3A06Z&sp=r&sv=2021-08-06&sr=b&rscd=inline&rsct=image/png&skoid=6aaadede-4fb3-4698-a8f6-684d7786b067&sktid=a48cca56-e6da-484e-a814-9c849652bcb3&skt=2024-04-17T22%3A32%3A23Z&ske=2024-04-18T22%3A32%3A23Z&sks=b&skv=2021-08-06&sig=AH5PvmftgbhhZHm1ZyCUtcUw%2BdboorxzV%2BDRUruIHVM%3D)

Please note, the link will expire after a certain duration, so be sure to view or download the image promptly.
agent run ID 0 finished!
Done

Complete agent
The most recent version is available on the GitHub repository.


```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.9;

// Uncomment this line to use console.log
// import "hardhat/console.sol";
import "./interfaces/IOracle.sol";

contract Agent {

    string public prompt;

    struct Message {
        string role;
        string content;
    }

    struct AgentRun {
        address owner;
        Message[] messages;
        uint responsesCount;
        uint8 max_iterations;
        bool is_finished;
    }

    mapping(uint => AgentRun) public agentRuns;
    uint private agentRunCount;

    event AgentRunCreated(address indexed owner, uint indexed runId);

    address private owner;
    address public oracleAddress;

    event OracleAddressUpdated(address indexed newOracleAddress);
```

```
    IOracle.OpenAiRequest private config;

    constructor(
        address initialOracleAddress,
        string memory systemPrompt
    ) {
        owner = msg.sender;
        oracleAddress = initialOracleAddress;
        prompt = systemPrompt;

        config = IOracle.OpenAiRequest({
            model : "gpt-4-turbo-preview",
            frequencyPenalty : 21, // > 20 for null
            logitBias : "", // empty str for null
            maxTokens : 1000, // 0 for null
            presencePenalty : 21, // > 20 for null
            responseFormat : "{\"type\":\"text\"}",
            seed : 0, // null
            stop : "", // null
            temperature : 10, // Example temperature (scaled up, 10 means 1.0),
> 20 means null
            topP : 101, // Percentage 0-100, > 100 means null
            tools : "[{\"type\":\"function\",\"function\":
{\"name\":\"web_search\",\"description\":\"Search the internet\",\"parameters\":
{\"type\":\"object\",\"properties\":{\"query\":
{\"type\":\"string\",\"description\":\"Search query\"}},\"required\":
[\"query\"]}}},{\"type\":\"function\",\"function\":
{\"name\":\"image_generation\",\"description\":\"Generates an image using Dalle-
2\",\"parameters\":{\"type\":\"object\",\"properties\":{\"prompt\":
{\"type\":\"string\",\"description\":\"Dalle-2 prompt to generate an
image\"}},\"required\":[\"prompt\"]}}}]",
            toolChoice : "auto", // "none" or "auto"
            user : "" // null
        });
    }

    modifier onlyOwner() {
        require(msg.sender == owner, "Caller is not owner");
        _;
    }

    modifier onlyOracle() {
        require(msg.sender == oracleAddress, "Caller is not oracle");
        _;
    }

    function setOracleAddress(address newOracleAddress) public onlyOwner {
        require(msg.sender == owner, "Caller is not the owner");
        oracleAddress = newOracleAddress;
        emit OracleAddressUpdated(newOracleAddress);
    }

    function runAgent(string memory query, uint8 max_iterations) public returns
(uint i) {
        AgentRun storage run = agentRuns[agentRunCount];

        run.owner = msg.sender;
        run.is_finished = false;
        run.responsesCount = 0;
        run.max_iterations = max_iterations;

        Message memory systemMessage;
        systemMessage.content = prompt;
```

```
        systemMessage.role = "system";
        run.messages.push(systemMessage);

        Message memory newMessage;
        newMessage.content = query;
        newMessage.role = "user";
        run.messages.push(newMessage);

        uint currentId = agentRunCount;
        agentRunCount = agentRunCount + 1;

        IOracle(oracleAddress).createOpenAiLlmCall(currentId, config);
        emit AgentRunCreated(run.owner, currentId);

        return currentId;
    }

    function onOracleOpenAiLlmResponse(
        uint runId,
        IOracle.OpenAiResponse memory response,
        string memory errorMessage
    ) public onlyOracle {
        AgentRun storage run = agentRuns[runId];

        if (!compareStrings(errorMessage, "")) {
            Message memory newMessage;
            newMessage.role = "assistant";
            newMessage.content = errorMessage;
            run.messages.push(newMessage);
            run.responsesCount++;
            run.is_finished = true;
            return;
        }
        if (run.responsesCount >= run.max_iterations) {
            run.is_finished = true;
            return;
        }
        if (!compareStrings(response.content, "")) {
            Message memory assistantMessage;
            assistantMessage.content = response.content;
            assistantMessage.role = "assistant";
            run.messages.push(assistantMessage);
            run.responsesCount++;
        }
        if (!compareStrings(response.functionName, "")) {
            IOracle(oracleAddress).createFunctionCall(runId,
response.functionName, response.functionArguments);
            return;
        }
        run.is_finished = true;
    }

    function onOracleFunctionResponse(
        uint runId,
        string memory response,
        string memory errorMessage
    ) public onlyOracle {
        AgentRun storage run = agentRuns[runId];
        require(
            !run.is_finished, "Run is finished"
        );
        string memory result = response;
        if (!compareStrings(errorMessage, "")) {
            result = errorMessage;
```

```
        }
        Message memory newMessage;
        newMessage.role = "user";
        newMessage.content = result;
        run.messages.push(newMessage);
        run.responsesCount++;
        IOracle(oracleAddress).createOpenAiLlmCall(runId, config);
    }

    function getMessageHistoryContents(uint agentId) public view returns
(string[] memory) {
        string[] memory messages = new string[]
(agentRuns[agentId].messages.length);
        for (uint i = 0; i < agentRuns[agentId].messages.length; i++) {
            messages[i] = agentRuns[agentId].messages[i].content;
        }
        return messages;
    }

    function getMessageHistoryRoles(uint agentId) public view returns (string[]
memory) {
        string[] memory roles = new string[]
(agentRuns[agentId].messages.length);
        for (uint i = 0; i < agentRuns[agentId].messages.length; i++) {
            roles[i] = agentRuns[agentId].messages[i].role;
        }
        return roles;
    }

    function isRunFinished(uint runId) public view returns (bool) {
        return agentRuns[runId].is_finished;
    }

    function compareStrings(string memory a, string memory b) private pure
returns (bool) {
        return (keccak256(abi.encodePacked((a))) ==
keccak256(abi.encodePacked((b))));
    }
}

Final run

Agent's task: Generate an image of current weather conditions in New York
Max iterations: 10
Task sent, tx hash:
0xd19bdd9539d5233ecef968319462362d8128eb2caabe2bf5df61977760e342e1
Agent started with task: "Generate an image of current weather conditions in New
York"
Created agent run ID: 1
STEP: You are a helpful assistant
STEP: Generate an image of current weather conditions in New York
STEP: [{"title": "Weather Forecast and Conditions for New York City, NY",
"link":
"https://weather.com/weather/today/l/96f2f84af9a5f5d452eb0574d4e4d8a840c71b05e22
264ebdc0056433a642c84", "snippet": "New York City, NY. As of 7:54 am EDT. 46\
u00b0. Drizzle. Day 49\u00b0 \u2022 Night 45\u00b0. Small Craft Advisory. Latest
News. Why 2024 Will Be One Of The Most Active Seasons ...", "position": 1},
{"title": "Hourly Weather Forecast for New York City, NY", "link":
"https://weather.com/weather/hourbyhour/l/f892433d7660da170347398eb8e3d722d8d362
fe7dd15af16ce88324e1b96e70", "snippet": "Wednesday, April 17 \u00b7 1 am. 57\
u00b0. 1%. Partly Cloudy. Feels Like57\u00b0. WindNNE 3 mph. Humidity46%. UV
Index0 of 11 \u00b7 2 am. 56\u00b0. 2%. Mostly Clear. Feels Like56\u00b0.",
"position": 2}, {"title": "Current Weather - New York, NY - AccuWeather",
"link": "https://www.accuweather.com/en/us/new-york/10021/current-weather/
```

349727", "snippet": "New York, NY \u00b7 Current Weather. 7:15 AM. 47\u00b0F.", "position": 3}, {"title": "Hourly Weather Forecast for Manhattan, NY", "link": "https://weather.com/weather/hourbyhour/l/Manhattan+NY?canonicalCityId=fc47c333c5d13e34e34c9fdb6e047ceb70f7891e01bc9e1d574b5f93f58aa76d", "snippet": "Friday, April 19 \u00b7 1 am. 47\u00b0. 4%. Mostly Cloudy. Feels Like45\u00b0. WindNE 5 mph. Humidity65% \u00b7 2 am. 47\u00b0. 4%. Mostly Cloudy. Feels Like45\u00b0. WindENE 5 mph. Humidity ...", "position": 4}, {"title": "New York, NY Weather Forecast - AccuWeather", "link": "https://www.accuweather.com/en/us/new-york/10021/weather-forecast/349727", "snippet": "10-Day Weather Forecast ; Today. 4/18. 50\u00b0 44\u00b0 ; Fri. 4/19. 56\u00b0 48\u00b0 ; Sat. 4/20. 67\u00b0 45\u00b0 ; Sun. 4/21. 60\u00b0 46\u00b0 ; Mon. 4/22. 65\u00b0 49\u00b0.", "sitelinks": [{"title": "Current Weather", "link": "https://www.accuweather.com/en/us/new-york/10021/current-weather/349727"}, {"title": "Daily", "link": "https://www.accuweather.com/en/us/new-york/10021/daily-weather-forecast/349727"}, {"title": "Hourly", "link": "https://www.accuweather.com/en/us/new-york/10021/hourly-weather-forecast/349727"}, {"title": "MinuteCast", "link": "https://www.accuweather.com/en/us/new-york/10021/minute-weather-forecast/349727"}], "position": 5}, {"title": "Weather Forecast and Conditions for Manhattan, NY", "link": "https://weather.com/weather/today/l/New+York+NY+USNY0996", "snippet": "Manhattan, NY. As of 11:01 am EDT. 61\u00b0. Partly Cloudy. Day 61\u00b0 \u2022 Night 48\u00b0. Small Craft Advisory. Latest News. 'Everything's Gone'- Tornado Survivor ...", "position": 6}, {"title": "Current WeatherNew York, NY - The Weather Network", "link": "https://www.theweathernetwork.com/en/city/us/new-york/new-york/current", "snippet": "Current WeatherNew York, NY. New York, NY. Updated 13 minutes ago. 9. \u00b0C. Light rain. Feels 6. H: 10\u00b0 L: 7\u00b0. Hourly. Full 72 hours \u00b7 6am. Cloudy. 9\u00b0.", "sitelinks": [{"title": "Hourly", "link": "https://www.theweathernetwork.com/us/hourly-weather-forecast/new-york/new-york"}, {"title": "14 Day", "link": "https://www.theweathernetwork.com/us/14-day-weather-trend/new-york/new-york"}, {"title": "7 Day", "link": "https://www.theweathernetwork.com/us/36-hour-weather-forecast/new-york/new-york"}], "position": 7}, {"title": "Accutrack Radar | New York Weather News - abc7NY", "link": "https://abc7ny.com/weather/", "snippet": "Today's Weather. New York City, NY. Current; Tonight; Tomorrow. CLOUDY. 50\u00b0. Feels Like. 45\u00b0. Sunrise. 6:13 AM. Humidity. 81%. Sunset. 7:38 PM. Windspeed. ENE ...", "sitelinks": [{"title": "Latest AccuWeather Forecast", "link": "https://abc7ny.com/nyc-weather-accuweather-forecast-new-york-city-tri-state/27410/"}, {"title": "AccuTrack Radar", "link": "https://abc7ny.com/weather/doppler/accutrack-radar/"}, {"title": "Weather Alerts", "link": "https://abc7ny.com/weather/alerts/"}, {"title": "New Jersey", "link": "https://abc7ny.com/weather/doppler/new-jersey/"}], "position": 8}, {"title": "Weather - NBC New York", "link": "https://www.nbcnewyork.com/weather/", "snippet": "Click to search by city or zip... New York City, NY. 47\u00b0. Light Drizzle. Feels like40\u00b0. Humidity90%. Precip0.09%. Wind16MPH. Temperature Precipitation. 47\u00ba.", "sitelinks": [{"title": "Maps and Radar", "link": "https://www.nbcnewyork.com/weather/maps/"}, {"title": "Weather", "link": "https://www.nbcnewyork.com/tag/weather/"}, {"title": "Weather Alerts", "link": "https://www.nbcnewyork.com/weather/severe-weather-alerts/"}, {"title": "School Closings", "link": "https://www.nbcnewyork.com/weather/school-closings/"}], "position": 9}, {"title": "New York City, NY Hourly Weather Forecast", "link": "https://www.wunderground.com/hourly/us/ny/new-york-city", "snippet": "New York City, NY Hourly Weather Forecaststar_ratehome ; 1:00 pm, Cloudy, 46 \u00b0F ; 2:00 pm, Cloudy, 47 \u00b0F ; 3:00 pm, Cloudy, 48 \u00b0F ; 4:00 pm, Cloudy, 48 \u00b0F ...", "position": 10}]
STEP: https://oaidalleapiprodscus.blob.core.windows.net/private/org-OIsLdcbHIc1F6swhIoWbkFsS/user-dvcNG9YUCv8srEVF1LN250bv/img-EniOzeTyLedEnBrZmo6jh0M7.png?st=2024-04-18T18%3A30%3A43Z&se=2024-04-18T20%3A30%3A43Z&sp=r&sv=2021-08-06&sr=b&rscd=inline&rsct=image/png&skoid=6aaadede-4fb3-4698-a8f6-684d7786b067&sktid=a48cca56-e6da-484e-a814-9c849652bcb3&skt=2024-04-17T22%3A37%3A08Z&ske=2024-04-18T22%3A37%3A08Z&sks=b&skv=2021-08-06&sig=PJUQuHsp52omwe6DXECOQK2M1pVj%2B9tW3LikbTkX2uE%3D

THOUGHT: I've generated an image representing the current weather conditions in New York as you requested. You can view the image by clicking the link below:

[View Image](https://oaidalleapiprodscus.blob.core.windows.net/private/org-OIsLdcbHIc1F6swhIoWbkFsS/user-dvcNG9YUCv8srEVF1LN250bv/img-EniOzeTyLedEnBrZmo6jh0M7.png?st=2024-04-18T18%3A30%3A43Z&se=2024-04-18T20%3A30%3A43Z&sp=r&sv=2021-08-06&sr=b&rscd=inline&rsct=image/png&skoid=6aaadede-4fb3-4698-a8f6-684d7786b067&sktid=a48cca56-e6da-484e-a814-9c849652bcb3&skt=2024-04-17T22%3A37%3A08Z&ske=2024-04-18T22%3A37%3A08Z&sks=b&skv=2021-08-06&sig=PJUQuHsp52omwe6DXECOQK2M1pVj%2B9tW3LikbTkX2uE%3D)
agent run ID 1 finished!
Done

What’s Next?
Congratulations on your own on-chain Agent! Explore further:

Implement a more advanced agent by adding retrieval-augmented generation to your contract.
Dive deeper into the Galadriel documentation, particularly the How It Works section, to understand the underlying technology.
Explore other Use Cases to get inspired for your next project.

Solidity reference
Overview
This page describes the details of the Solidity interface to the oracle, and how to use it in your contract. We assume you are familiar with the high-level architecture of Galadriel.

Oracle interface
The oracle is exposed as a contract that you can call from your own contract. It exposes two types of functionality: calling LLMs and calling external tools.

Any time you make a call to the oracle, you need to do at least two things:

Call a method on the oracle contract to create a new call (to a model or a tool).
Implement a callback method on your contract that the oracle will call once the response is available.
In each page of this reference you’ll see the method you need to call and the corresponding callback method you need to implement.

See source code for all the methods in the ChatOracle contract.

See the Usage examples below on how to integrate depending on your use case.

Usage examples

Calling an LLM
For calling an LLM, one option is to use the basic createLlmCall method, which has a simple interface but little configurability. The other option is to use the more complex but more powerful createOpenAiLlmCall method — to make a call to OpenAI-provided LLMs — or the createGroqLlmCall method — to make a call to Groq-provided LLMs.

Using the createLlmCall example: to call an LLM you need to:

1
Call the createLlmCall method on the oracle contract. This will create a new LLM

call.

2
Implement two methods into your contract that the oracle will call to get the message history necessary for the LLM, including the system prompt: getMessageHistoryContents and getMessageHistoryRoles.

3
Implement the onOracleLlmResponse method into your contract. This method will be called when the LLM result is available (in a separate transaction potentially tens of seconds later).

Using the other two methods is analogous, but you need to use the corresponding callback method:

createLlmCall -> implement callback onOracleLlmResponse - see Basic LLM usage
createOpenAiLlmCall -> implement callback onOracleOpenAiLlmResponse - see OpenAI
createGroqLlmCall -> implement callback onOracleGroqLlmResponse - see Groq

Calling an external tool
If you are calling an external tool, you need to:

1
Call the createFunctionCall method on the oracle contract. This will create a new tool call.

2
Implement the onOracleFunctionResponse method into your contract. This method will be called when the tool result is available (in a separate transaction potentially tens of seconds later).

Since method calls do not reference a message history, you don’t need to implement any history getter methods.

Examples
Please refer to the example contracts, corresponding to the outlined use cases, for complete usage examples.

Basic

createLlmCall
This is a simplified function that method makes a call to OpenAI GPT-4 with default parameters and provides no configurability. If you want to use other models, or configure other parameters like temperature, see createOpenAiLlmCall.

Arguments

promptCallbackId
uint
Identifier which will be passed into the callback.

Returns

counter
uint
An internal counter of the oracle, which is incremented on every call.

onOracleLlmResponse
Called when the result of an LLM call, created with createLlmCall, is available.

Arguments

callbackId
uint
The identifier you passed into the createLlmCall method.

response
string
The result of the LLM call.

errorMessage
string
An error message. Contains an empty string if there was no error, and a description of the error otherwise.

OpenAI

createOpenAiLlmCall
This is a function that calls an OpenAI LLM. You can configure almost any parameter that can be configured in the Create chat completion OpenAI endpoint.

Arguments

promptCallbackId
uint
Identifier which will be passed into the callback.

config
OpenAiRequest
An object that contains all configuration parameters for the LLM call. See below for details on the OpenAiRequest object.

Returns

counter
uint
An internal counter of the oracle, which is incremented on every call.

OpenAiRequest object
The OpenAiRequest object (see source in IOracle.sol) has the following fields:

model
string
The identifier for the model, gpt-4-turbo, gpt-4-turbo-preview or gpt-3.5-turbo-1106.

frequencyPenalty
int8
An integer between -20 and 20, mapped to a float between -2.0 and 2.0; values greater than 20 are treated as null.

logitBias
string
A JSON string representing logit bias or an empty string if none.

maxTokens
uint32
The maximum number of tokens to generate, with 0 indicating null.

presencePenalty

int8
An integer between -20 and 20, mapped to a float between -2.0 and 2.0; values greater than 20 are treated as null.

responseFormat
string
A JSON string specifying the format of the response or an empty string if default.

seed
uint
A seed for the random number generator, with 0 indicating null.

stop
string
A string specifying stop sequences, or an empty string to indicate null.

temperature
uint
A temperature setting for randomness, with values from 0 to 20; values greater than 20 indicate null.

topP
uint
Controls diversity via nucleus sampling, with values from 0 to 100 percent; values greater than 100 indicate null.

tools
string
A JSON list of tools in OpenAI format, or an empty string for null, where names must match supported tools.

toolChoice
string
Specifies tool selection strategy, with values “none”, “auto”, or an empty string which defaults to “auto” on OpenAI’s side.

user
string
Identifier for the user making the request.

onOracleOpenAiLlmResponse
Called when the result of an LLM call, created with createOpenAiLlmCall, is available.

Arguments

callbackId
uint
The identifier you passed into the createOpenAiLlmCall method.

response
OpenAiResponse
The result of the LLM call. See below for details on the OpenAiResponse object.

errorMessage
string
An error message. Contains an empty string if there was no error, and a description of the error otherwise.

OpenAiResponse object
The OpenAiResponse object (see source in ChatOracle.sol) has the following

fields, most of which map directly to the fields in the OpenAI API response:

id
string
Unique identifier for the completion, generated by OpenAI.

content
string
The model output. Either this field or functionArguments and functionName will be populated, depending on whether the output is a normal message or a function call.

functionName
string
Name of the function invoked, if any.

functionArguments
string
Arguments passed to the function, if any, as a JSON string. For format details, refer to the specific function documentation.

created
uint64
Timestamp of creation.

model
string
The model name used for generation.

systemFingerprint
string
Identifier for the system generating the completion.

object
string
Type of the object, always chat.completion.

completionTokens
uint32
Number of tokens generated in the completion.

promptTokens
uint32
Number of tokens in the prompt.

totalTokens
uint32
Total number of tokens, including both prompt and completion.

Groq

createGroqLlmCall
This is a function that calls an open-source LLM hosted by Groq. You can configure almost any parameter that can be configured in the Create chat completion Groq endpoint.

Arguments

promptCallbackId
uint
Identifier which will be passed into the callback.

config
GroqRequest
An object that contains all configuration parameters for the LLM call. See below for details on the GroqRequest object.

Returns

counter
uint
An internal counter of the oracle, which is incremented on every call.

GroqRequest object
The GroqRequest object (see source in ChatOracle.sol) has the following fields:

model
string
The model to be used, options are llama-3.1-70b-versatile, llama-3.1-8b-instant, llama3-8b-8192, llama3-70b-8192, mixtral-8x7b-32768, or gemma-7b-it. See Groq docs for more info on the models.

frequencyPenalty
int8
Penalty for frequency, int -20 to 20, mapped to float -2.0 to 2.0. Values greater than 20 result in null.

logitBias
string
JSON string specifying logit bias, or an empty string for none.

maxTokens
uint32
Maximum number of tokens to generate, with 0 indicating null.

presencePenalty
int8
Penalty for presence, int -20 to 20, mapped to float -2.0 to 2.0. Values greater than 20 result in null.

responseFormat
string
JSON string indicating the format of the response, or an empty string for default format.

seed
uint
Seed for random number generation, with 0 indicating null.

stop
string
Stop sequence(s) for generation, or an empty string to indicate none.

temperature
uint
Temperature for generation, 0 to 20, with values greater than 20 indicating null.

topP
uint
Top p sampling parameter, 0 to 100 percentage, with values greater than 100 indicating null.

user
string

User identifier or context.

onOracleGroqLlmResponse
Called when the result of an LLM call, created with createGroqLlmCall, is available.

Arguments

callbackId
uint
The identifier you passed into the createGroqLlmCall method.

response
GroqResponse
The result of the LLM call. See below for details on the GroqResponse object.

errorMessage
string
An error message. Contains an empty string if there was no error, and a description of the error otherwise.

GroqResponse object
The GroqResponse object (see source in ChatOracle.sol) has the following fields, most of which map directly to the fields in the Groq API response:

id
string
Unique identifier for the completion, generated by Groq.

content
string
The content associated with the completion.

created
uint64
Timestamp of creation.

model
string
The model name used for the completion.

systemFingerprint
string
System identifier that generated the completion.

object
string
Type of the object, always chat.completion.

completionTokens
uint32
Number of tokens in the completion.

promptTokens
uint32
Number of tokens in the prompt.

totalTokens
uint32
Total number of tokens, sum of prompt and completion tokens.

Message history
The two methods in combination — getMessageHistoryContents and getMessageHistoryRoles — are used to provide the oracle with the message history necessary for the LLM call. They are called by the oracle after you invoke createLlmcall or an analogous method for other LLM providers.

The methods should each return a list, one with message contents and the other listing the roles of the message authors. The list lengths should be equal for a given callbackId.

For multimodal models such as the gpt-4-turbo, which can handle complex queries including those with image URLs, a more integrated approach is used. This is facilitated through the getMessageHistory method, which provides both the message roles and contents in a structured format suitable for processing by these models.

Try calling out gpt-4-turbo with image URl using chat_vision.ts script.

For example, if the message history is the following:

role content
system You are a helpful assistant
user Hello!
assistant Hi! How can I help?
user How big is the Sun?
Then getMessageHistoryContents should return the following list of 4 items:

```
[
    "You are a helpful assistant",
    "Hello!",
    "Hi! How can I help?",
    "How big is the Sun?"
]
```

…and getMessageHistoryRoles should return the following list of 4 items:

```
["system", "user", "assistant", "user"]
```

The signatures of these methods should be:

getMessageHistoryRoles
Arguments

callbackId
uint
The identifier you passed into the createLlmCall method.

Returns

return
string[]
A list of roles. Each role in the list is a string and one of system, user, or assistant.

getMessageHistoryContents
Arguments

callbackId
uint
The identifier you passed into the createLlmCall method.

Returns

return
string[]
A list of message contents.

getMessageHistory
For multimodal models like gpt-4-turbo, the getMessageHistory method is used to fetch a combined structure of messages, which includes both the content and the role within a single response. This method simplifies handling more complex data formats and supports enhanced features like image processing.

Arguments

callbackId
uint
The identifier you passed into the createLlmCall method.

Returns

return
IOracle.Message[]
An array of IOracle.Message, where each message structure contains:

role
string
The role of the message sender (e.g., system, user, assistant).
content
IOracle.Content[]
An array of Content, where each Content includes:
contentType
string
text or image_url, specifies whether the content is text or an image.
value
string
The actual message or image URL.

Tools

createFunctionCall
This method makes a call to a tool — see options below. For every tool the input and output are of string type, and you choose a tool by specifying the functionType parameter.

Arguments

functionCallbackId
uint
Identifier which will be passed into the callback.

functionType
string
Specifies which function to call. Must be one of: image_generation, web_search.

functionInput
string
Input to the function.

Returns

counter
uint
An internal counter of the oracle, which is incremented on every call.

The currently available tools are:

Image generation

Web search

Code interpreter

onOracleFunctionResponse
Called when the result of a tool call, created with createFunctionCall, is available.

Arguments

callbackId
uint
The identifier you passed into the createFunctionCall method.

response
string
The result of the tool call.

errorMessage
string
An error message. Contains an empty string if there was no error, and a description of the error otherwise.

RAG
You can add a knowledge base to your LLM app or agent. A knowledge base is broadly useful for including lots of information in the context — information that is too long to always include in the prompt.

This technique, called Retrieval Augmented Generation (RAG), is a combination of a retriever and a generator. The retriever finds the relevant information from the knowledge base, and the generator uses this information to generate a response. In our case, the generator is an LLM, and as explained below, you can use a vector database through the Galadriel oracle as the retriever.

Before you can call any of the steps above, you need to:

Extract text from your documents (PDFs, books, websites, etc).
Upload the chunks to IPFS.
Request the oracle to index it.
We have provided a script that does this for you — follow the steps in this README; this is a prerequisite for your contract to be able to call the below methods.

After indexing your documents, you should have an IPFS CID representing your knowledge base (the above script provides this).

createKnowledgeBaseQuery
This method makes a call to query the knowledge base. The oracle will return the documents that match the query in the callback: see onOracleKnowledgeBaseQueryResponse.

Arguments

kbQueryCallbackId
uint
Identifier which will be passed into the callback.

cid
string
IPFS CID representing the indexed knowledge base, which will be queried.

query
string
The query to be used to retrieve the documents from the knowledge base.

num_documents
uint32
Maximum number of documents to retrieve.

Returns

counter
uint
An internal counter of the oracle, which is incremented on every call.

onOracleKnowledgeBaseQueryResponse
Called when the result of a knowledge base query, created with createKnowledgeBaseQuery, is ready.

Arguments

kbQueryCallbackId
uint
The identifier you passed into the createKnowledgeBaseQuery method.

documents
string[]
A list of document contents that were retrieved from the knowledge base.

errorMessage
string
An error message. Contains an empty string if there was no error, and a description of the error otherwise.

Oracle address
The current address of the teeML oracle on Galadriel devnet is:

0x68EC9556830AD097D661Df2557FBCeC166a0A075

This is the oracle provided by the Galadriel team, running in a TEE.

If you wish, you can set up your own: see here.

If the above oracle is not responding, you may want to use a secondary oracle in the development process. This secondary oracle does not run in a TEE. The address of that oracle is:

0x0352b37E5680E324E804B5A6e1AddF0A064E201D

FAQ

FAQ for not getting answers back from oracle

1. Is the correct oracle address whitelisted?
The correct oracle address is available here:
https://docs.galadriel.com/quickstart

2. Do you have funds on your address?
Can check from the explorer:
https://explorer.galadriel.com/address/my_wallet_address

3. Is your contract deployed?
Can you see your contract in here:
https://explorer.galadriel.com/address/my_contract_address

4. Can you get an answer with the quickstart?
Did you try out the service with the quickstart tutorial?
https://docs.galadriel.com/quickstart

5. Was the tx sent to the oracle?
Check the explorer for your own account:
https://explorer.galadriel.com/address/my_wallet_address

Do you see a successful tx?

6. Is your Solidity code working as expected?
Have you tested your code? Can you send back answers to your own contract?

If your code has an error in the callback function and the tx gets reverted then the Oracle just can’t send the response back. Make sure you yourself can call this function on your contract:
https://github.com/galadriel-ai/contracts/blob/45bf72707adc21c5b2d3a2f95ac337ff4dde006c/contracts/contracts/interfaces/IChatGpt.sol#L13

Make sure one of the expected interfaces is implemented
https://docs.galadriel.com/reference/overview

7. Can the oracle get the chat history from your contract?
Can you call out getMessageHistory functions? For example this:
https://github.com/galadriel-ai/contracts/blob/45bf72707adc21c5b2d3a2f95ac337ff4dde006c/contracts/contracts/interfaces/IChatGpt.sol#L25

8. Can I run the e2e tests on my contract?
Galadriel team has provided a test suite that you can run on your LLM based contract. You can find it here:
https://github.com/galadriel-ai/contracts/tree/main/contracts#contract-debugging

Oracle address

For users
Setting up a wallet
Galadriel should be compatible with any wallet that supports EVM. Our recommendation is to use MetaMask.

Whenever you attempt to connect your wallet to a dApp running on Galadriel network, you should be automatically prompted to add the Galadriel network to your app. Alternatively, you could connect manually as described below.

Galadriel Devnet is occasionally reset from genesis, so you may need to reset your network; see instructions below.

Network reset

If you previously connected to the Devnet and it has since been reset from genesis, you need to migrate to the new network.

To migrate your wallet, reset your account activity in your wallet. If you are using Metamask, follow these instructions.

To migrate your application, make sure you are using the correct RPC URL (devnet.galadriel.com) and Oracle address, and reset off-chain nonce handling if using any.

Connect manually

To connect to the Galadriel devnet manually, follow the guide to add a custom network RPC and use the following settings:

Network Name: Galadriel Devnet
New RPC URL: https://devnet.galadriel.com
Chain ID: 696969
Currency Symbol: GAL
Block Explorer URL: https://explorer.galadriel.com

For users

Faucet

To make any transactions on our devnet, you need devnet tokens. These are used only for the purpose of development and have no real value.

Currently, the only way to receive those is via Discord. This is done to prevent abuse and make sure every developer has access to devnet tokens.

The faucet currently dispenses 1 GAL, allowing for about 150-200 calls to the oracle.

To get devnet tokens:

1
Join Discord

Join our Discord server and go to the #devnet-faucet channel.

2
Post request

Post a message in the #devnet-faucet channel following this exact format, replacing <address> with your EVM wallet address.

!faucet <address>

In a few seconds, you should receive the devnet tokens in your wallet.

For users

Block explorer

Currently the only block explorer on Galadriel is explorer.galadriel.com, provided by the Galadriel team.

The explorer also hosts a powerful REST API providing indexed data: see the API

section in the explorer sidebar.

The block explorer allows you to do the following and more.

View transactions: See individual transactions included in each block.
Explore blocks: Browse and inspect individual blocks, including details like block height, size, and timestamp.
Search: Search for specific transactions, blocks, addresses, or tokens using their unique identifiers.
See network statistics: Access real-time and historical data on the network.
Decode raw data: Interpret raw transaction or block data.